VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Paritní hry: struktura, algoritmy, experimenty

Parity Games: Structure, Algorithms, Experiments

2018 Bc. Lukáš Žák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Lukáš Žák**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Paritní hry: struktura, algoritmy, experimenty
Parity Games: Structure, Algorithms, Experiments

Jazyk vypracování: čeština

Zásady pro vypracování:

Paritní hry jsou jednoduše formulované hry na ohodnocených grafech, které úzce souvisejí s problémy automatizovaného ověřování korektnosti programů. K řešení paritních her byly navrženy různé algoritmy, není ale stále jasné, zda problém je polynomiální. Účelem diplomové práce je návrh a implementace softwarového nástroje, inspirovaného existujícími podobnými nástroji, který by umožnil snadné experimentování s variantami vybraných řešicích algoritmů.

1. Seznamte se s problémem paritních her a otevřenou otázkou existence polynomiálních algoritmů k jejich řešení (z literatury konzultované s vedoucím práce).
2. Seznamte se s projektem PGSolver autorů Friedmann a Lange: https://github.com/tcsprojects/pgsolver a s dalšími zdroji konzultovanými s vedoucím práce.
3. Navrhněte softwarový nástroj, který by umožnil snadné experimentování s různými variantami algoritmů, diskutovanými s vedoucím práce.
4. Nástroj ve zvoleném prostředí implementujte a proveďte příslušné experimenty.
5. Výsledky zhodnoťte a osvětlete případné nové poznatky.

Seznam doporučené odborné literatury:

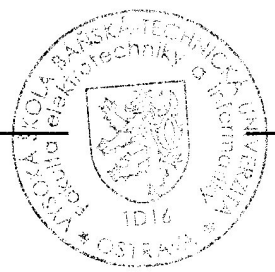
Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. RNDr. Petr Jančar, CSc.**

Datum zadání: 01.09.2015

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. června 2018 .....

Rád bych poděkoval panu prof. RNDr. Petr Jančarovi, CSc. za vedení mé práce, za jeho věcné připomínky, rady, názory, trpělivost a především za jeho čas a podporu, kterou mi poskytl při sepisování dané problematiky.

Abstrakt

Diplomová práce se zabývá tématem paritních her. V první části je práce zaměřená na vysvětlení a vymezení paritních her a všech pojmů s tím spojených. Jedná se hlavně o to, co jsou to paritní hry a jaké jsou jejich pravidla. Dále jsou popsány důležité struktury, jako je graf, na kterém se odehrávají a pojmy používané v řešících algoritmech. Hlavní částí je následné rozebrání konkrétních algoritmů. Ty jsou zde popsány a definovány včetně příkladů ilustrujících jejich průběh. Na to navazuje popis aplikace vzniklé v rámci této práce. Aplikace využívá uvedených algoritmů k řešení paritních her v přehledném uživatelském rozhraní s možností testování daných algoritmů. Poslední část se zabývá otestováním této aplikace a porovnáním jednotlivých implementovaných algoritmů.

Klíčová slova: Paritní hry, Rekurzivní algoritmus, Hledání malých dominií, Small progress measures, Strategy Improvement, C#, PGSolver

Abstract

The thesis deals with the issue of parity games. In the first part of the work we focus on explaining and defining parity games and all of the terms associated with it. It is mainly about what parity games are and how are they played. Right after follows description of the important structures, such as a graph, on which they are set and terms used in solving algorithms. The main part of the thesis is explaining specific algorithms dealing with parity games. These are described and defined, including examples of how each algorithm works. This is followed by a description of the application resulting in this work. The application uses the specified algorithms for solving parity games in intuitive user interface with options for testing them. The last part deals with testing of the algorithms and comparing the results of algorithms.

Key Words: Parity games, Recursive algorithm, Small dominion finder, Small progres measures, Strategy Improvement, C#, PGSolver

Obsah

Seznam použitých zkratek a symbolů

SAT	–	Boolean satisfiability problem
EBNF	–	Extended Backus–Naur Form (Rozvinutá Backusova–Naurova forma)
GUI	–	Grafické uživatelské rozhraní
LINQ	–	Language Integrated Query
SPM	–	Small progress measures
IDE	–	Vývojové prostředí

Seznam obrázků

Seznam tabulek

1 Úvod

Diplomová práce se zabývá otázkou paritních her. Paritní hry jsou jednoduché hry hrané na ohodnocených grafech dvěma hráči, jejichž průběh je nekonečný. I přes svá jednoduchá pravidla není stále zřejmé, zda je lze řešit v polynomiálním čase. Řešení paritních her je důležité také z důvodu jejich převoditelnosti na jiné typy nekonečných her.

Paritní hry mohou například sloužit jako prostředek pro automatickou verifikaci systémů. Další využití mohou nalézt při automatickém návrhu systémů. Při návrhu systému představují vrcholy hry žádané chování systému. Hráči jsou pak považováni za systém a vnější okolí systému. Výsledek paritní hry následně slouží jako návrh systému, který splňuje modelované chování. Převoditelnost paritních her na jiné druhy her se využívá například u problémů spojených s μ-calculus model checking.

Úvod práce (Kapitola 2) je věnován popisu paritních her. Nejprve jsou paritní hry definovány, což zahrnuje definování grafu hry, hráčů a pravidel, kterými se hra řídí. Definice paritních her je zakončena popisem výherní podmínky, která určuje výherce hry. Kapitola dále pokračuje definicí a vysvětlením pojmů, které jsou využívány v následujících kapitolách věnujících se algoritmům pro řešení paritních her.

Kapitola 3 je zaměřena na představení a definování algoritmů, které byly vyvinuty pro řešení paritních her. U každého popisovaného algoritmu je uveden jeho teoretický základ a vysvětlen princip fungování. První z popisovaných algoritmů je nejstarší a je nazýván rekurzivní algoritmus.[3, 5] Jeho autorem je Wiesław Zielonka a i přes svou exponenciální složitost je považován za jeden z nejlepších. Druhým algoritmem je algoritmus hledání malých dominií.[6, 7] Je to deterministický subexponenciální algoritmus představený M. Jurdzińskim, který funkčně navazuje na rekurzivní algoritmus. Hlavní ideou tohoto algoritmu je kombinace modifikovaného rekurzivního algoritmu a předzpracování grafu hry. Kapitola pokračuje představením zástupce iterativního řešení paritních her. Třetím algoritmem je algoritmus nazvaný Small Progress Measures.[8] Algoritmus vrcholům přiřadí vektory, které určují jeho kvalitu a iterativně všechny vektory upravuje. Poslední popisovaný algoritmus je nazván Discrete Strategy Improvement.[9] Také se jedná o iterativní algoritmus, který postupně mění strategii vrcholů a snaží se dosáhnout optima. Po popisu každého algoritmu následuje krátký příklad, jež ilustruje průběh popsaného algoritmu.

Pro řešení paritních her existuje softwarový nástroj, který se jmenuje PGSolver.[11] Tento nástroj byl vyvinut na univerzitě v Mnichově. V Kapitole 4 je PGSolver stručně představen a také jsou uvedeny jeho možnosti a způsob použití. Na konci kapitoly je popsán textový formát definice hry, který PGSolver využívá.

Kapitola 5 popisuje návrh a zpracování aplikace vyvinuté v rámci této práce. Na úvod obsahuje stručný nástin toho, jak by měla aplikace vypadat a její funkční možnosti. Aplikace je vytvořena pro systém Windows a umožňuje řešit paritní hry popsanými algoritmy. Mimo řešení her umožňuje její druhá část i testování algoritmů. Vše je v podobě aplikace s přehledným

grafickým uživatelským rozhraním. Dále práce pokračuje popisem některých implementačních detailů. Následující Kapitola 6 je podána jako uživatelská příručka. Popisuje jednotlivé části aplikace a jejich ovládání.

V závěru práce jsou uvedeny výsledky získané během testování implementovaných algoritmů ve vytvořené aplikaci. Testování je prováděno na předpřipraveném setu her.[14] Set her obsahuje jak náhodné hry, tak některé speciální, na nichž určité algoritmy vykazují exponenciální složitost. Posledním bodem je uvedení srovnání s nástrojem PGSolver.

Závěr diplomové práce je věnován shrnutí nabytých poznatků a finálnímu zhodnocení výsledků práce.

2 Úvod do paritních her

Paritní hry jsou nekonečné hry s nulovým součtem a dokonalou informací hrané dvěma hráči na konečném, ohodnoceném a orientovaném grafu. Hra probíhá tak, že hráči pohybují sdíleným tokenem po hranách grafu z jednoho vrcholu do druhého. Posuny tokenu vytváří nekonečnou sekvenci po sobě jdoucích vrcholů. Jednotlivé posuny tokenu nazýváme tah, sekvenci tahů pro daný graf nazýváme partie.

2.1 Paritní hry

Paritní hry se odehrávají na orientovaném grafu a definujeme je jako:

Definice 1 *Mějme paritní hru $G = (V, E, \lambda)$ takovou, že $V = V_0 \cup V_1$ je disjunktní sjednocení množiny vrcholů V_0 a V_1 a vrcholy V_0 patří hráči 0 a vrcholy V_1 patří hráči 1. Dále obsahuje množinu orientovaných hran $E \subseteq V \times V$. Funkce λ každému vrcholu V přiřadí jeho prioritu jako přirozené číslo $\lambda : V \to \mathbb{N}$.*[1]

Graf, na kterém je hra hrána nesmí obsahovat žádné listové vrcholy a proto z každého vrcholu grafu musí vystupovat alespoň jedna hrana do nějakého následníka. Každý vrchol $v \in V$ má alespoň jednu výstupní hranu do nějakého následníka $n \in V$, tj. musí platit, že $\forall v \in V \; \exists n \in V : (v, n) \in E$.

Definice 2 *Následníka vrcholu $v \in V$ označíme vrchol v', pro který platí $vEv' = \{v' \in V \mid (v, v') \in E\}$. Předchůdce vrcholu definujeme jako $v'Ev = \{v' \in V \mid (v', v) \in E\}$.*

Takto definovanou podobu herního grafu si můžeme prohlédnout na Obrázku 1, který obsahuje vizualizaci, kterou budeme dále používat. Každý vrchol má v sobě vepsanou prioritu, která mu byla přiřazena funkcí λ.

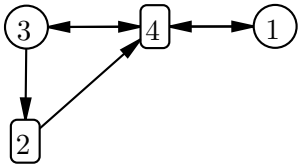
2.2 Hráči

Hru hrají vždy dva hráči, které značíme hráč 0 a hráč 1. Vrcholy na Obrázku 1 vyznačené kolečkem patří hráči 0 a vrcholy ztvárněné čtverečkem patří hráči 1. Pro rozlišení hráče, který je aktuálně na tahu zavedeme značení σ, kde $\sigma \in \{0, 1\}$. Protihráče v daném kontextu značíme $\overline{\sigma}$ s tím, že platí $\overline{\sigma} = 1 - \sigma$.

2.3 Průběh hry

Hra začíná umístěním tokenu (můžeme si ho představit jako herní figurku) na libovolný vrchol grafu. Tento vrchol označíme jako počáteční v_0. Pokud je vrchol $v_0 \in V_0$, a tedy patří hráči 0 pak hráč 0 začíná hru. V opačném případě vrchol patří do $v_0 \in V_1$ a hru začíná hráč 1. Obecně

[1] Různé vrcholy grafu mohou mít přiřazenou i stejnou prioritu.



Obrázek 1: Jednoduchý graf paritní hry

je na tahu hráč, který je vlastníkem vrcholu, na kterém se aktuálně nalézá token. Hráč, který je na tahu přesune token z vrcholu v do některého z jeho následníků $v' \in vE$. Obecně je-li token položen na vrchol $v \in V_\sigma$, tak hráč σ přesune token do nějakého následníka $v' \in vE$. Dále hra pokračuje a token je na nějakém vrcholu grafu $v' \in V_\sigma$. Hráč σ opět přesune token z vrcholu v' do nějakého následníka $v'' \in v'E$. Tento postup je opakován nekonečně krát a vzniká nekonečná hra.

Definice 3 *Partie paritní hry je nekonečná v případě, že z daného počátečního vrcholu skládáme postupnými tahy nekonečně dlouhou posloupnost po sobě následujících vrcholů. Vytváříme tak posloupnost $\pi = v_0 v_1 \ldots \in V^\omega$, kde každý vrchol $v_{i+1} \in v_i E$ a pro všechny i platí, že $i \in \mathbb{N}$.*

2.4 Určení výherce hry

Výhra v paritní hře záleží na tom, ze kterého vrcholu se v grafu hry začíná hrát. Z každého vrcholu hry vyhrává právě jeden z hráčů. Proběhnou-li partie ze všech vrcholů grafu, vznikne nám pro každý vrchol sekvence π, podle které dokážeme určit, zda z daného vrcholu vyhrává hráč 0 nebo hráč 1. Výhru hráče 0 v paritní hře G určujeme takto:

Definice 4 *Výhru hráče 0 na vrcholu v_0 definujeme tak, že v nekonečné posloupnosti vrcholů π začínající v počátečním vrcholu v_0 určíme vrchol s maximální prioritu opakujícího se nekonečně krát a určíme jeho paritu. Pokud je parita tohoto vrcholu sudá vyhrává hráč 0. Formálně zapíšeme takto: nechť $Inf(\pi)$ je množina priorit vrcholů opakujících se v π nekonečně krát často, pak:*

$$Inf(\pi) = \{k \ \in \mathbb{N} | \forall j \in \mathbb{N} : \exists i \in \mathbb{N} : i > j \ a \ k \ = \lambda(v_i)\}$$

Hráč 0 vyhrává hru s posloupností vrcholů π pouze tehdy, když maximum $Inf(\pi)$ je sudé.

Obdobně můžeme zavést pravidla pro výhru hráče 1.

Definice 5 *Paritní hru G začínající ve vrcholu v_0 vyhrává hráč 1 jestliže jsou odehrány tahy π a $Inf(\pi)$ je liché.*

Pro paritní hry existuje také duální výherní podmínka. Opět se hledá nekonečná sekvence vrcholů π s tím rozdílem, že výherce partie se neurčuje podle maximální priority, ale podle minimální priority. Podmínka na paritu takového vrcholu se nemění. Pro tyto dvě verze vyhodnocování vítěze

se můžeme setkat se značením max-parity a min-parity games. Oba tyto druhy vyhodnocování jsou ekvivalentní a můžeme je mezi sebou jednoduše převádět.

Definice 6 *Hru G můžeme převést na hru G' s opačnou výherní podmínkou tak, že určíme d jako nejvyšší prioritu v G. Pokud je tato priorita sudá, tak d = maximální priorita. V případě, že je lichá tak d = maximální priorita + 1. Poté pro každý vrchol provedeme $\lambda(v) = d - \lambda(v)$.*

Tímto způsobem můžeme převádět hru z min-parity na max-parity game a naopak.

2.5 Řešení paritních her

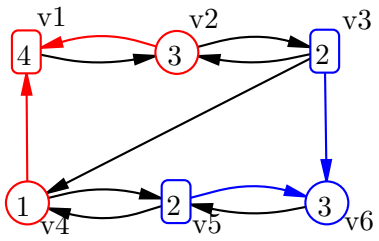
Řešení paritních her spočívá v určení výherních vrcholů, ze kterých dokáže vyhrát hráč 0, a ze kterých hráč 1. K tomu, aby hráč vyhrál má určenu svou výherní strategii. Určení výherních strategií jednotlivých hráčů je sekundárním řešením paritních her. V této sekci definujeme tyto a další pojmy potřebné v algoritmech řešících paritní hry.

2.5.1 Strategie

Strategie v paritních hrách určuje, jak bude hráč na daném vrcholu dále táhnout tokenem. Jednoduše řečeno, strategie pro hráče σ určuje, kterou hranou by se měl z aktuálního vrcholu vydat dále. Strategie určuje hranu $(v_k, v_{k+1}) \in E$, která obecně záleží na nějaké dosavadní konečné cestě grafem $\pi = v_0, \ldots, v_k$. Hráč by měl při vybírání svého následujícího tahu uvažovat celou dosavadní historii hry. V případě paritních her to není potřeba a pro výběr následujícího tahu nám stačí tzv. poziční strategie. Nemusíme znát celou historii hry, ale záleží pouze na aktuálním vrcholu.

Strategie je částečná funkce $f_\sigma : V^*V_\sigma \to V$. Průběh hry $\pi = v_0 v_1 \ldots v_l$ je v souladu s funkcí f_σ právě tehdy, když pro všechny i v intervalu $0 \leq i < l$ a $v_i \in V_\sigma$ je funkce f_σ definována na $v_0 \ldots v_i$ a následující vrchol $v_{i+1} = f_\sigma(v_0 \ldots v_i)$. Funkce f_σ jako svůj argument přijímá celou dosavadní posloupnost odehraných vrcholů π. Každá partie π je v souladu s funkcí f_σ právě tehdy, když i každý její prefix je v souladu s touto funkcí. Funkci f_σ nazýváme strategie hráče σ na $U \subseteq V$.

Poziční (bezpaměťová, memoryless) strategie je speciální případ strategie, kdy nezáleží na dosavadním průběhu hry, ale pouze na posledním dosaženém vrcholu. Výběr hrany, přes kterou se bude pokračovat do dalšího vrcholu $(v_k, v_{k+1}) \in E$ záleží pouze na v_k. Je to tedy funkce $f_\sigma : V_\sigma \to V$. Když hra dospěje do nějakého vrcholu $v \in V_\sigma$, hráč σ vybírá následující hranu podle $f_\sigma(v)$. Pokud mají hráči určené své poziční strategie, tak mají jednoznačně určený tah z každého svého vrcholu. Tento tah provedou vždy bez ohledu na historii hry. Jejich strategie se tak v průběhu hry nemění.



Obrázek 2: Paritní hra o 6 vrcholech v vyznačením vítězných vrcholů hráčů

2.5.2 Výherní strategie a výherní regiony

Strategii f_σ nazýváme výherní strategie pro hráče σ a nějaký vrchol $v \in V$, pokud tato strategie zajistí hráči σ výhru každé partie začínající z vrcholu v, bez ohledu na to, jak hraje jeho protihráč. Jinak řečeno, hráč σ vyhraje paritní hru G začínající ve vrcholu v, pokud má tento hráč výherní strategii pro hru G z v.

Řešením paritní hry G jsou dvě množiny vrcholů, a to $W_0, W_1 \subseteq V$. Množina W_0 obsahuje veškeré vrcholy, ze kterých je hráč 0 schopen vyhrát hru G, pokud začíná v některém z těchto vrcholů a použije svou výherní strategii. Pro množinu W_1 pak platí, že z daných vrcholů vyhrává hráč 1. Pro každý vrchol v hry platí, že můžeme určit, zda $v \in W_0$ nebo $v \in W_1$ a také, že $W_0 \cap W_1 = \emptyset$. Tyto dvě množiny nazýváme výherní regiony.

Problémem řešení paritních her je určení množin výherních vrcholů W_0 a W_1. Sekundárním problémem je určit pro všechny vrcholy z obou množin jejich výherní (poziční) strategie, a tím určit výherní strategie pro oba z hráčů.[3]

Pro ukázku výše popsaných termínů uvedeme Příklad 1, který popisuje hru na 6 vrcholech a určíme, jaké vrcholy jsou vítězné pro kterého hráče.

Příklad 1

Na Obrázku 2 vidíme příklad paritní hry o 6 vrcholech označených $v_1 \dots v_6$. Zadání hry bylo převzato z literatury.[10] V každém vrcholu je vepsáno číslo, které reprezentuje prioritu daného vrcholu. Každý vrchol přísluší jednomu z hráčů. Vrcholy obarvené červeně (v_1, v_2, v_4) jsou výherní pro hráče 0, ty obarvené modře (v_3, v_5, v_6) jsou výherní pro hráče 1. Obarvené hrany značí výherní strategie hráčů.

Hráč 0 se snaží donutit protihráče k tomu, aby byl co nejčastěji odehrán tah do vrcholu s prioritou 4 (v_1), jelikož je to vrchol s nejvyšší sudou prioritou. Navíc hráč 1 nemá strategii, aby v_1 nebyl zahrán nekonečně krát a má jedinou možnost hrát do vrcholu s prioritou 3 (v_2), který vlastní hráč 0. Tím vznikne nekonečná sekvence vrcholů s prioritou 3, 4. Hráč 0 z vrcholu v_2 bude vždy hrát zpět do vrcholu v_1. Nejvyšší priorita je tak sudá a vítězí hráč 0. Z vrcholu v_4 bude hráč 0 vždy táhnout do vrcholu $v1$ a situace se opakuje.

Obdobná situace platí pro vítězné vrcholy hráče 1, který se snaží udržet hru v cyklu vrcholů s prioritou 3 a 2 (v_5 a v_6). Jakmile se hra dostane do jednoho z vrcholů v_3 nebo v_5, tak bude

hráč 1 vždy hrát do vrcholu v_6 odkud již hráč 0 nemůže uniknout a nejvyšší priorita objevující se nekonečně krát bude 3 vrcholu v_6 a zajistí výhru hráči 1.

Lze pozorovat, že se hra rozpadla na dvě části, ze kterých vyhrává vždy jeden z hráčů. Tyto dvě množiny vrcholů jsme označili jako výherní regiony. Výherní region hráče 0 vypadá následovně: $W_0 = \{v_1, v_2, v_4\}$. Zbylé vrcholy patří do W_1. Výherní strategie hráčů má význam určovat pouze u těch vrcholů, ze kterých daný hráč vyhrává a zároveň je vlastní. U hráče 0 na obrázku vidíme vyznačenou výherní strategii z vrcholů v_2 a v_4.

■

2.5.3 Vyjádření nekonečné hry jako konečné

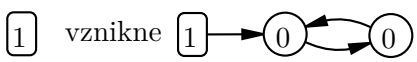
Každá nekonečná hra se dá vyjádřit jako hra konečná. Je to zapříčiněno tím, že se zabýváme pouze hrami na grafu s konečným počtem vrcholů a tím, že hráči mají nasazeny své výherní strategie. Stručně nastíníme, jak z nekonečné hry vyjádřit hru konečnou, a jaký to má vliv na určení výherních regionů.

Každou partii hry G začínající v nějakém vrcholu v_0 si můžeme rozepsat pomocí stromové struktury. Kořenem stromu je počáteční vrchol v_0. Strom se dále větví tak, že nové uzly vznikají podle toho, jaké následníky má aktuální vrchol. Pro příklad si vezmeme počáteční vrchol mající dvě výstupní hrany a tedy 2 následníky. Připojením následníků ke kořeni stromu začneme skládat všechny možné cesty grafem z v_0. Opakováním postupu na nově přidané vrcholy do stromu vznikají další a další úrovně. Takhle vzniklý strom je nekonečný a reprezentuje všechny možné postupy π hrou z vrcholu v_0.

Tím, že se hra odehrává na konečném grafu, vzniknou v každém průběhu π vzniklého stromu opakující se podstromy. Tyto podstromy se ve stromě nadále opakují a značí cyklus v grafu hry. To si můžeme představit tak, že listový vrchol, který je shodný s kořenovým vrcholem podstromu, bychom spolu propojili za vzniku cyklu. Nalezením těchto opakujících se podstromů a jejich odstraněním ukončíme nekonečný rozvoj stromu reprezentující hru a přeneseně vznikne konečný průchod hrou. Odstraněním opakujících se podstromů vznikla konečná cesta stromem, u které není obtížné určit, který z hráčů by vyhrál při dané posloupnosti navštívených vrcholů. Výherce je určen modle maximální priority objevující se v opakujícím se podstromu. Veškerý prefix tohoto podstromu je z pohledu řešení nepodstatný, protože nekonečně krát opakující se priority obsahuje právě daný podstrom. Opakující se podstrom, neboli cyklus v průběhu π nazveme lichý, jestliže nejvyšší priorita v něm objevující se je lichá a sudý, když je maximální priorita sudá.

Zároveň se jedná o návod na nedeterministický polynomiální algoritmus k řešení paritních her. Algoritmus by nedeterministicky hádal v každém vrcholu strategii daného hráče až po nějaký listový vrchol (začátek daného cyklu) a pak jednoduše polynomiálně ověřil, zda daný cyklus je sudý nebo lichý, a tím ověřil, jestli je daná strategie pro určitého hráče výherní či nikoliv. Strategie se stává výherní, pokud alespoň jedna cesta stromem je pro hráče výherní.

Obrázek 3: Transformace dead endu vrcholu Hráče 0



Obrázek 4: Transformace dead endu vrcholu Hráče 1

Mějme případ, že oba hráči již mají nasazenu svou poziční strategii pro své vrcholy. Tím je jednoznačně dán následovník každého vrcholu. Vytvořením stromu pro nějaký vrchol získáme pouze jednu větev značící průběh hry za použití daných pozičních strategií. Tato větev je ukončena v okamžiku průchodu již navštíveným vrcholem. Tento cyklus vznikne nejpozději po n krocích, pokud n značí počet vrcholů grafu. Zjištění výherce takové hry je triviálním ověřením sudosti vzniklého cyklu z každého vrcholu při použití daných strategií. Lze také pozorovat, že z daného vrcholu může vyhrát vždy právě jeden z hráčů. Říkáme pak, že paritní hry jsou determinované. Každý z vrcholů hry G patří právě do jedné množiny výherních vrcholů W_0 nebo W_1.

2.5.4 Dead end a převod do tvaru bez dead endů

V grafu hry se mohou objevit vrcholy bez odchozí hrany, takzvané dead endy. Dle definice hry graf nesmí dead endy obsahovat. Tyto vrcholy mohou být na obtíž pro obecné řešení některých algoritmů např. Small Progress Measure. Dead endy zmiňujeme z důvodu možnosti chybného zadání hry do aplikace vyvíjené v této práci. Proto uvedeme transformace hry na hru bez dead endů. Tento postup nastíněný v literatuře řeší tento převod s lineární časovou a logaritmickou prostorovou složitostí.[2]

V grafu se mohou vyskytovat také vrcholy s hranou sama na sebe. Tyto vrcholy nejsou pro správné řešení kritické a transformace je uvedena pouze pro úplnost.

Uvedené transformace převedou hru G na ekvivalentní hru G' již bez dead endů, případně bez smyček.

- Dead end

  Transformaci pro hráče 0 a hráče 1 vidíme na Obrázku 3 a Obrázku 4. Vlastník vrcholu by neměl kam táhnout a prohrál by. Přidají se tedy 2 soupeřovy vrcholy s jemu prospěšnou prioritou, které mezi sebou tvoří smyčku a z původního vrcholu vyvedeme do této smyčky hranu.

- Vrchol se smyčkou sama na sebe

  Pokud je priorita vrcholu prospěšná vlastníkovi vrcholu, hráč by vytvářel nekonečnou sekvenci tohoto vrcholu. Přidáním nových vrcholů patřících tomuto hráči, s jemu prospěšnou

Obrázek 5: Transformace smyčky na vrcholu Hráče 0

Obrázek 6: Transformace smyčky na vrcholu Hráče 1

prioritou a přidáním hran mezi nimi, odstraníme tuto smyčku sama na sebe. Ilustrace principu je na Obrázku 5 a Obrázku 6.

Pokud priorita vrcholu není prospěšná vlastníkovi můžeme tuto hranu odstranit. Zůstane-li upravený vrchol bez odchozí hrany, postupujeme stejně jako u dead endu.

2.5.5 Uzavřená množina a σ past

Uzavřená množina je taková množina vrcholů $U \subseteq V$, že když jednou hra zavítá do některého z vrcholu této množiny, může si hráč svou strategií vynutit, aby již zůstala v této množině. Množina U je σ-uzavřená právě tehdy, když všechny vrcholy $v \in U$ hráče σ mají alespoň jednu hranu vedoucí do U a všechny vrcholy $v \in U$ hráče $\overline{\sigma}$ mají všechny odchozí hrany směřující do U.

Definice 7 *Mějme množinu $U \subseteq V$. U můžeme nazvat σ-uzavřenou právě tehdy, když $\forall v \in U \cap V_\sigma \; \exists u \in U : vE$ a zároveň $\forall v \in U \cap V_{\overline{\sigma}} \; \forall u \in vE : u \in U$.*

S uzavřenou množinou úzce souvisí pojem σ-past. Množinu vrcholů $U \in V$ nazýváme σ-past pokud platí následující: hráč σ ze svých vrcholů v U nemá hranu, díky které by vyvedl hru z množiny U. A zároveň hráč $\overline{\sigma}$ má ze všech svých vrcholů v U alespoň jednu hranu vedoucí do U.

Definice 8 *Množinu $U \subseteq V$ nazveme σ-past právě tehdy, když $\forall v \in U \cap V_{\overline{\sigma}} \; \exists u \in U : vE$ a zároveň $\forall v \in U \cap V_\sigma \; \forall u \in vE : u \in U$.*

2.5.6 Dominium

Dominium nazýváme množinu vrcholů $D \subseteq V$, která je uzavřená a všechny její vrcholy jsou vyhrávány právě jedním z hráčů. Výhra nastává bez ohledu na to, jak hraje protihráč.

Definice 9 *Množina vrcholů $D \subseteq V$ je σ-dominium právě tehdy, když je σ-uzavřená a pro $\forall v \in D \cap V_\sigma$ má hráč σ vítěznou strategii.*

2.5.7 Atraktor

Atraktor se nazývá taková množina vrcholů A, ve které má hráč strategii takovou, že dokáže hru udržet v A a dosáhnout v konečném počtu kroků nějaké žádoucí množiny vrcholů X.

Definice 10 *Mějme paritní hru $G = (V, E, \lambda)$, množinu $X \subseteq V$, které chceme dosáhnout, hráče $\sigma \in \{0, 1\}$ a značení $Attr^i_\sigma$. Dolní index označuje hráče, pro nějž se atraktor vytváří a horní index maximální počet kroků nutných k dosažení množiny X. Atraktor je definován následovně:*

$$Attr^0_\sigma(G, X) = X$$

Je přirozeně vidět, že toto platí. Pokud chceme dosáhnout množiny X v 0 krocích musíme se již ve vrcholu této množiny nacházet. Zvětšením maximálního počtu kroků k dosažení atraktoru na 1 získáme následující definici.

$$Attr^1_\sigma(G, X) = \{v \in V_\sigma | vE \cap X \neq \emptyset\} \cup \{v \in V_{\overline{\sigma}} | vE \subseteq X\}$$

Induktivním krokem pak vytváříme

$$Attr^{i+1}_\sigma(G, X) = Attr^i_\sigma(G, X) \cup Attr^1_\sigma(G, Attr^i_\sigma(G, X)) \text{ pro všechny } i > 1$$

Výsledný σ-atraktor získáme tehdy, když i je nejnižší možné pro které platí, že

$$Attr^{i+1}_\sigma(G, X) = Attr^i_\sigma(G, X)$$

Takové i vždy existuje, protože se hry odehrávají na konečném grafu. Výsledným atraktorem je sjednocení jednotlivých atraktorů přes všechny i.

$$Attr_\sigma(G, X) = \bigcup_{i \in \mathbb{N}} attr^i_\sigma(G, X)$$

Příklad 2

Hra na Obrázku 7 nyní poslouží jako příklad pro ukázku tvorby atraktoru. Prioritu vepsanou ve vrcholech budeme pro zjednodušení považovat zároveň za identifikátor vrcholu. Úkolem je vytvořit atraktor k vrcholu 1 hráče 1. V prvním iteraci získáme v atraktoru pouze zkoumaný vrchol:

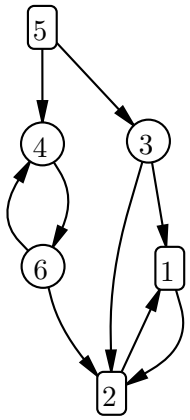
$$attr^0_1(G, 1) = \{1\}$$

V následující iteraci přidáme všechny vrcholy hráče 1, které mají alespoň jednu výstupní hranu do vrcholu 1 a všechny vrcholy hráče 0, které mají všechny výstupní hrany do vrcholu 1. Podmínku splňuje pouze vrchol 2 a vznikne:

$$attr^1_1(G, attr^0_1(G, 1)) = attr^0_1 \cup \{2\}$$

Postup opakujeme dokud můžeme přidávat další vrcholy.

$$attr^2_1(G, attr^1_1(G, attr^0_1(G, 1))) = attr^0_1 \cup attr^1_1 \cup \{3\}$$



Obrázek 7: Paritní hra pro ukázku tvorby atraktorů

$$attr_1^3(G, attr_1^2(G, attr_1^1(G, attr_1^0(G, 1)))) = attr_1^0 \cup attr_1^1 \cup attr_1^2 \cup \{5\}$$

Vrcholy 4 a 6 mají spojující hrany pouze mezi sebou a nemáme tedy možnost je nijak zahrnout do atraktoru. Výsledný atraktor vrcholu 1 pro hráče 1 je:

$$attr_1(G, 1) = \{1, 2, 3, 5\}$$

■

2.5.8 Podhry

Podhra G' vznikne odebráním některých vrcholů z originální hry G společně se vstupními a výstupními hranami. Po odstranění nějakého vrcholu z G musí G' zůstat spojitá a každý vrchol musí mít alespoň jednu výstupní hranu.

Definice 11 *Mějme hru $G = (V, E, \lambda)$ a nějakou podmnožinu vrcholů $U \subseteq V$. Podhra $G' = (V', E', \lambda)$ vznikne odstraněním všech vrcholů množiny U a všech příslušejících hran ze hry G značeno $G' = G \backslash U$. Za podmínky, že hra G' je stále správně vytvořená paritní hra.*

2.6 Složitost paritních her

Pro paritní hry do dnešní doby stále není znám žádný polynomiální algoritmus. Je to jeden z mála přirozených problémů, které spadají do třídy složitosti $NP \cap co-NP$. Marcin Jurdziński také dokázal, že redukcí paritních her na mean payoff games lze dosáhnout třídy složitosti $UP \cap co-UP$.[4] To znamená, že hry lze řešit jednoznačným nedeterministickým polynomiálním Turingovým strojem. Paritní hry jsou převoditelné v polynomiálním čase na další typy her jako mean payoff games, simple stochastic games a discounted payoff games.

3 Algoritmy řešící paritní hry

Pro paritní hry vznikaly a vznikají různé typy algoritmů. Nicméně stále ještě není znám žádný algoritmus, který by dokázal paritní hry řešit v polynomiálním čase. Jeden z prvních algoritmů byl rekurzivní algoritmus. Následoval algoritmus vycházející přímo z rekurzivního nazvaný hledání malých dominií. Objevily se také odlišné přístupy k řešení. Jedním z nich je pomocí zlepšování strategií, kdy hráči zlepšují své strategie do doby, než vylepšit nejdou. V neposlední řadě pak small progress measure, kdy se modifikují vrcholům přiřazené vektory čísel, dokud nedosáhnout rovnovážného stavu. Existuje mnoho článků popisujících nové a nové algoritmy, ale v principu se vždy jedná o nepatrnou modifikaci jednoho ze zmíněných algoritmů nebo úpravu (omezení) definice vstupní hry.[2]

3.1 Rekurzivní algoritmus

Rekurzivní algoritmus pro řešení paritních her vychází z práce Wiesława Zielonky.[3, 5] V principu se jedná o algoritmus typu rozděl a panuj, kdy dochází k postupnému rozdělování hry na menší části, které se rekurzivně řeší až do triviálního případu hry o nula vrcholech. V případě, že dosáhneme hry s nulovým počtem vrcholů získáváme automaticky prázdné výherní regiony. Kompletní výherní regiony následně vychází z postupného skládání řešení z redukovaných podher.

Algoritmus vychází z toho, že vyšší priority vrcholů mají ve hře větší váhu a pro hráče je výhodné snažit se hrát do vrcholů s vyššími prioritami. Na začátku řešení hry zjistíme nejvyšší prioritu d vyskytující se ve hře G tak, že $d = max\{\lambda(v)|v \in V\}$ a vytvoříme neprázdnou množinu vrcholů U mající prioritu d, tedy $U = \{v|\lambda(v) = d\}$. Z důvodu následného vytváření atraktorů určíme paritu priority d jako $\sigma = d \ mod \ 2$.[3]

Jako první krok vytvoříme σ-atraktor A k množině U. Tuto získanou množinu odstraníme z grafu hry G a získáme menší podhru $G' = G \backslash A$. Pokud hráč σ vyhrává celou hru G', pak také vyhrává celou hru G. Je to dáno tím, že pokud hráč $\overline{\sigma}$ bude hrát do nějakého z vrcholů A, vítězná strategie hráče σ bude hrát do U a zajistit si tak nejvyšší prioritu d vyskytující se ve hře nekonečně krát. Pokud by se hráč $\overline{\sigma}$ rozhodl setrvat svými tahy v G', tak je z předpokladu dáno, že vyhrává hráč σ.

Pokud hráč σ podhru G' nevyhrává celou a hráč $\overline{\sigma}$ vyhrává alespoň z nějakých vrcholů v G' přiřadíme mu neprázdný výherní region $W'_{\overline{\sigma}} \neq \emptyset$. O těchto vrcholech víme, že jsou pro hráče $\overline{\sigma}$ výherní i v původní hře G. Hráč σ není schopen přinutit hráče $\overline{\sigma}$ opustit $W'_{\overline{\sigma}}$ a vynutit si hru do jím vyhrávajících vrcholů. Vypočítáme tedy nový $\overline{\sigma}$-atraktor B množiny $W'_{\overline{\sigma}}$ v původní hře

[2] V největší míře se objevují úpravy konstant u hledání dominií nebo definování speciálních případů her s omezením počtu vrcholů, výstupních hran nebo priorit.

[3] Paritu si pro zjednodušení označíme σ stejně jako značíme hráče. Důvodem je, že budeme vytvářet atraktory pro hráče odpovídající paritě maximální priority.

G, odstraníme tento atraktor ze hry a vrcholy z $W'_{\overline{\sigma}}$ označíme jako výherní pro $\overline{\sigma}$. Následně se začne rekurzivně řešit vzniklá podhra $G \backslash B$.

Tento postup můžeme vyjádřit pseudokódem na Výpisu 1.

```
Vyres(G):
   if V == ∅:
      (W_0, W_1)=(∅, ∅)
   else:
      d = max priorita v G
      U = {v∈V | λ(v)=d}
      σ = d % 2
      σ̄= 1 - σ
      A = Attr_σ(U)
      (W'_0, W'_1) = Vyres(G\A)
      if W'_σ̄ == ∅:
         W_σ = W'_σ ∪ A
         W_σ̄ = ∅
      else:
         B = Attr_σ̄(W'_σ̄)
         (W'_0, W'_1) = Vyres(G\B)
         W_σ = W'_σ
         W_σ̄ = W'_σ̄ ∪ B
return (W_0, W_1)
```

Výpis 1: Pseudokód rekurzivního algoritmu

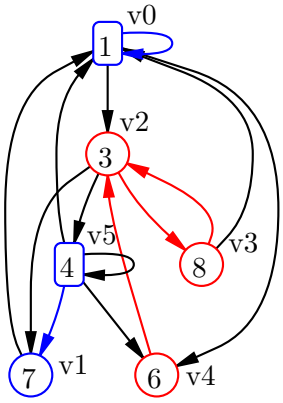
Důkaz toho, že se algoritmus zastaví a vrátí výherní vrcholy hráčů si můžeme rozepsat. Mějme hru G s počtem vrcholů $|V|$. Pokud je G prázdná hra algoritmus končí a vrátí správné výsledky.

Indukcí budeme uvažovat hru, kde $|V| > 0$ s parametry d = max. priorita, $\sigma = d \bmod 2$ a množinou vrcholů U s prioritou d.

Vytvořením atraktoru A k U a rekurzivním řešením hry $G' = G \setminus A$ je jisté, že $|V_{G'}| < |V|$. Indukční předpoklad pak zaručuje, že algoritmus skončí a vrátí výherní regiony W_0 a W_1.

Pokud $W'_{\overline{\sigma}} = \emptyset$ tvrdíme, že hráč σ vyhrává vrcholy W'_σ i v původní hře. Pokud bude hra probíhat v W'_σ víme z předpokladu, že na těchto vrcholech vyhrává hráč σ. Pokud hra v nějakém okamžiku navštíví jakýkoliv z vrcholů atraktoru A tak si σ vynutí nekonečně krát navštívení vrcholu z množiny U. Nejvyšší priorita tak bude d. V tomto okamžiku výpočet končí a vrací odpovídající výherní regiony.

V případě, že $W'_{\overline{\sigma}} \neq \emptyset$, vytvoříme k těmto vrcholům $\overline{\sigma}$-atraktor B a rekurzivně řešíme podhru $G'' = G \setminus B$. Počet vrcholů hry se sníží takže $|V_{G''}| < |V|$. Podle indukčního předpokladu algoritmus skončí a vrací W''_0, W''_1.



Obrázek 8: Řešení rekurzivním algoritmem

Pokud je W''_σ výherní v G, tak protihráč z něj nemůže uniknout od svého výherního regionu a nemůže uniknout ani do B jinak by takový vrchol byl zahrnut v B. U hráče $\overline{\sigma}$ je výherní region $W''_{\overline{\sigma}}$. Pokud hra zůstane v tomto regionu je podle předpokladu jisté, že na těchto vrcholech vyhraje. Pokud hra navštíví nějaký z vrcholů atraktoru B, tento tah zapříčiní navštěvování $W'_{\overline{\sigma}}$, a protože se jedná o výherní region vzhledem k G' a hráč σ nemůže navštívit A (protože se jedná o σ atraktor), vítězem vrcholů je podle předpokladu $\overline{\sigma}$. Algoritmus korektně skončí a vrátí výherní regiony hráčů.

V nejhorším případě nám každé rekurzivní volání ze hry odstraní právě 1 vrchol a v každém volání mohou být až 2 další rekurzivní volání. Hloubka zanoření tak může být až $n = |V|$. Výsledná výpočetní složitost rekurzivního algoritmu je $O(2^n)$.

Příklad 3

Pro ilustraci principu rekurzivního algoritmu vyřešíme příklad paritní hry vyobrazené na Obrázku 8. Hra se skládá z 6 vrcholů označených $v_0 \dots v_5$. Na obrázku je hra již vyřešena a výherní vrcholy jsou označeny barevně. Červené vrcholy patří do výherního regionu hráče 0, modré hráče 1.

První krok je určení nejvyšší priority. Nejvyšší prioritu má vrchol v_3 a proto $d = 8$. Následuje určení množiny $U = \{v_3\}$. Z d určíme $\sigma = 8 \bmod 2 = 0$. Tato priorita je prospěšná pro hráče 0. Podle definice atraktoru vytvoříme 0-atraktor k U. Atraktor bude na začátku obsahovat samotný vrchol v_3. V první iteraci vytváření přibude vrchol v_2 a v druhé iteraci vrchol v_4. Tím vznikne množina $A = \{v_3, v_2, v_4\}$. Vrcholy A odstraníme ze hry a rekurzivně řešíme zmenšenou podhru.

Maximální priorita v podhře je 7. Vytvořením 1-atraktoru k vrcholu v_1 získáme $A = \{v_1, v_5\}$. Odstraněním A dále redukujeme hru. Pro další rekurzi vznikla podhra o jednom vrcholu. Nejvyšší priorita je 1 vrcholu v_0. Atraktor nemůže přidat žádné další vrcholy a obsahuje pouze v_0. Odstraněním dostaneme prázdnou hru a v dalším rekurzivním volání je splněna podmínka ukončení rekurze. Funkce vrací prázdné výherní regiony pro oba hráče.

Při návratu z rekurze probíhá pro každou podhru kontrola vrácených výherních regionů. Pokud je výherní region $W_{\overline{\sigma}}$ prázdný, tak hráč σ vyhrává ze všech vrcholů atraktoru A. Všechny

vrcholy atraktoru A přidáme do W_σ a $W_{\overline{\sigma}} = \emptyset$.

V prvním návratu z rekurze jsou oba výherní regiony prázdné. $W_0 = \emptyset$ splňuje podmínku pro přiřazení vrcholů atraktoru hráči 1, vznikne $W_1 = \{v_0\}$ a funkce vrací tento výherní region společně s $W_0 = \emptyset$. Po druhém návratu z rekurze má hráč 0 nadále prázdný výherní region. Vrcholy aktuálního atraktoru přiřadíme jako výherní hráči 1 a vrátíme výherní regiony $W_0 = \emptyset$, $W_1 = \{v_0, v_1, v_5\}$.

Pokud při kontrole $W_{\overline{\sigma}} \neq \emptyset$, hráč $\overline{\sigma}$ vyhrává z $W_{\overline{\sigma}}$. Vytvoříme k těmto vrcholům atraktor B jako $Attr_{\overline{\sigma}}(W_{\overline{\sigma}})$, jehož vrcholy odstraníme z grafu a spustíme rekurzivní řešení na vzniklé podhře. Výherní regiony vrácené při návratu z rekurze rozdělíme tak, že $W_\sigma = W_\sigma$ a $W_{\overline{\sigma}} = W_{\overline{\sigma}} \cup B$.

Nyní se algoritmus nachází na první úrovni rekurze, $\sigma = 0$, $W_0 = \emptyset$ a $W_1 = \{v_0, v_1, v_5\}$. Hráč $\overline{\sigma}$ nemá prázdný výherní region. Vytvoříme 1-attraktor $B = Attr_1(W_1)$, odstraníme jej ze hry a rekurzivně řešíme zbylou hru. Atraktor B nepojme žádné nové vrcholy - jedná se o uzavřenou množinu. Vrcholy B jsou bezpečně výherní pro hráče 1. Ve zbylé hře je $d = 8$ a $\sigma = 0$. Vytvořením 0-atraktoru $Attr_0(v_3)$ získáme $A = \{v_2, v_3, v_4\}$. Po odstranění A, řešíme v rekurzivním volání prázdnou hru a funkce vrací prázdné výherní regiony. Výherní region hráče 1 je při návratu prázdný a hráč 0 získává všechny vrcholy A. Funkce se vrátila zpět na první úroveň rekurze. Přiřazením výsledku z posledního volání získáme $W_0 = \{v_2, v_3, v_4\}$ a $W_1 = \{v_0, v_1, v_5\}$. Končí první úroveň rekurze a funkce vrací výherní regiony obou hráčů.

■

3.2 Hledání malých dominií

Algoritmus hledání malých dominií patří do kategorie modifikací rekurzivního algoritmu představeného v předchozí části. Hlavní motivací pro vývoj bylo zlepšení složitosti v nejhorším případě. Zlepšení spočívá ve spojení rekurzivního algoritmu a hledání dominií hrubou silou.[6]

Základem algoritmu je předzpracování grafu hry před rekurzivním voláním. Předzpracování spočívá v tom, že se v grafu hledají dominia do předem dané velikosti. Nalezení dominia podle Definice 2.5.6 znamená, že tyto vrcholy grafu tvoří uzavřenou množinu a příslušný hráč má pro všechny vítěznou strategii. Pokud je dominium hledané velikosti ve hře nalezeno, vytvoří se k němu atraktor a atraktor se odstraní ze hry. Vrcholy atraktoru jsou přidány do výherního regionu hráče a rekurzivně se řeší podhra. Pokud dominium není nalezeno využívá se pro řešení upravený rekurzivní algoritmus. Pro lepší představu je ve Výpise 2 uveden pseudokód.

Řešení hry začíná určením počtu vrcholů hry a parametru velikosti dominií l. Parametr l je kritický pro určení velikosti hledaných dominií. Na hodnotě tohoto parametru závisí funkce `dominion`. Funkce `dominion` má jako argumenty graf hry a velikost hledaných dominií a provádí předzpracování grafu hry hrubou silou. Funkce postupně generuje všechny podmnožiny vrcholů hry až do velikosti určené parametrem l. Těchto podmnožin vrcholů je až $\binom{n}{l}$, kde $n = |V|$.

```
VyresSDominii(G):
n = |V|
if n == 0
   return(∅,∅)
l = √n
(D,σ) = dominion(G,l)
σ̄ = 1 - σ
G′ = G\Attr_σ(D)
if D == ∅
   (W_0, W_1) = VyresRekurzive(G)
else:
      (W′_0, W′_1) = VyresSDominii(G′)
      (W_σ, W_σ̄) = (W′_σ ∪ Attr_σ(D), W′_σ̄)
return (W_0, W_1)
```

Výpis 2: Pseudokód algoritmu hledání dominií.

U každé vygenerované podmnožiny je zapotřebí zkontrolovat, zda se jedná o uzavřenou množinu. Pokud zkoumaná množina není uzavřená, nemůže být dominiem. Takovou množinu ignorujeme a vygenerujeme další množinu pro zkoumání. Zjistíme-li, že množina je σ-uzavřená, ověříme ji, zda je také dominiem. Dominium poznáme tak, že hráč σ vyhrává ze všech vrcholů. Splňuje-li množina podmínky dominia, vracíme její vrcholy jako návratovou hodnotu D společně s vyhrávajícím hráčem σ a končíme generování podmnožin vrcholů. Pokud funkce vygeneruje všechny kombinace vrcholů a nenalezneme mezi nimi dominium, vrací funkce prázdnou množinu.

Parametr l značně ovlivňuje složitost hledání dominií. Nastavení l jako příliš malé zajistí rychlé generování a hledání dominií, ale bude potřeba více rekurzivních volání. Naopak nastavení l jako moc velké může způsobit příliš náročné generování a ověřování všech podmnožin grafu. Bylo zjištěno a je doporučováno hodnotu l volit jako $l = \left\lceil \sqrt{|V|} \right\rceil$, případně $l = \sqrt{2|V|}$. V nejhorším případě je složitost hledání dominií $O(n^l)$.

Pokud funkce `dominion` nalezla ve hře dominium, vytvoříme σ-atraktor k dominiu $Attr_\sigma(D)$. Vytvořený atraktor je odstraněn ze hry a rekurzivně se zavolá řešení zbylé podhry s počtem vrcholů $|V'| < |V|$. V ideálním případě složíme výherní regiony při návratu z rekurzí tak, že atraktor přiřadíme do W_σ.

Nenalezneme-li žádné dominium, řešení se zavede do upraveného rekurzivního algoritmu. Rekurzivní algoritmus je upraven tak, že jeho vnitřní rekurzivní volání jsou nahrazena voláním funkce hledání malých dominií.

Algoritmus v nejhorším možném případě vykazuje složitost $n^{O(\lceil\sqrt{n}\rceil)}$, při zvoleném parametru $l = (\lceil\sqrt{n}\rceil)$.[7]

Příklad 4

Pro příklad řešení poslouží hra z Obrázku 8. Na začátku řešení určíme velikost hry a spočítáme parametr $l = 3$. Funkce `dominion` generuje postupně podmnožiny vrcholů od velikosti 1 do 3. První nalezené dominium obsahuje vrchol v_0 a není těžké ověřit, že se jedná o dominium.

Obsahuje pouze jeden vrchol hráče 1. Prvně ověříme, zda je množina uzavřená. Vrchol má pouze jednu výstupní hranu zpět na sebe a splňuje podmínku na to být 1-uzavřená. Vrchol bereme jako správně vytvořenou hru a ověříme, zda hráč 1 tuho hru vyhrává ze všech vrcholů. Hru vyřešíme a zjistíme, že ze všech vrcholů vyhrává hráč 1. Vrchol v_0 v podhře je dominium, protože splňuje podmínku uzavřenosti a výhernosti. Funkce `dominion` vrací vrchol v_0 a vyhrávajícího hráče 1. K dominiu vytvoříme atraktor $Attr_1(v_0)$, který zahrne vrcholy v_1 a v_5. Vytvořený atraktor je bezpečně výherní pro hráče 1.

Odstraněním atraktoru v původní hře získáme hru o třech vrcholech. Rekurzivně se řeší zbylá podhra. Ve hře zbyly 3 vrcholy a parametr $l = 2$. Funkce `dominion` nenalezne žádné dominium o jednom vrcholu a generuje větší podmnožiny. Podmnožina s vrcholy v_2 a v_3 tvoří 0-uzavřenou množinu a jsou výherní pro hráče 0. Vytvořením 0-atraktoru $Attr_0(\{v_2, v_3\})$ zahrneme i poslední vrchol v_4. Vytvořený atraktor je bezpečně výherní pro hráče 0. Odstraněním atraktoru dostaneme prázdnou hru.

Následné rekurzivním volání proběhne na prázdné hře a funkce vrací prázdné výherní regiony. Postupným návratem z rekurzivních volání skládáme výsledné výherní regiony. Zjištěné výherní regiony jsou $W_0 = \{v_2, v_3, v_4\}$ a $W_1 = \{v_0, v_1, v_5\}$. ∎

3.3 Small progress measures

Algoritmus Small progress measures (dále jen SPM) řeší paritní hry parciálně. Algoritmus je schopen v jednom průběhu získat výherní vrcholy a příslušné strategie pouze pro hráče 0. Výherní vrcholy hráče 1 se určí jako $W_1 = G \setminus W_0$. V návrhu algoritmu se Jurdziński omezuje na paritní hry s minimální výherní podmínkou. V dalším popisu budeme pracovat s touto ekvivalentní verzí hry.[8]

Algoritmus všem vrcholům hry přiřadí vektor přirozených čísel, který nazýváme progress measure. Tyto vektory udržují informaci o stavu vrcholu vůči jeho následníkům a indikují výskyt sudých cyklů ve hře. Algoritmus v cyklu prochází všechny vrcholy a v každém průchodu porovnává vektor vrcholu s vektory jeho následníků. Pokud vektor vrcholu nesplňuje určité podmínky vůči vektorům jeho následníků, tak je příslušně upraven. Iterace algoritmu probíhají, dokud jsou vektory vrcholů upravovány. Jakmile dosáhnou vektory rovnovážného stavu, výpočet končí a z hodnot vektorů určíme výherní region hráče 0. Získání výherních strategií spočívá v porovnání vektorů následníků a vybrání následníka s nejlepším hodnocením.

Definice 12 *Mějme paritní hru G, index hry d = maximální priorita ve hře. Small progress measure $\rho(v)$ je vektor přirozených čísel $M_1 \times \ldots \times M_d$, kde $M_i = 0$, pokud je i sudé nebo*

$M_i = n$, kde $n \in \{0, 1, \ldots, m_i\}$, pokud je liché. Hodnota m_i je dána jako počet vrcholů, kde $\lambda(v) = i$.

Funkce $\rho : V \rightarrow M_0 \times \ldots \times M_d$ přijímá jako vstup vrchol a vrací small progress measure vektor.

Pro účely porovnávání zavedeme pojem mezní progress measure. Tento vektor ohraničuje shora všechny ostatní a reprezentuje maximální možnou hodnotu.

Definice 13 *Mezní progress measure vektor je vektor přirozených čísel $M_1 \times \ldots \times M_d$, kde $M_i = 0$, pokud je i sudé, nebo $M_i = m_i$, pokud je liché. Hodnota m_i je dána jako počet vrcholů, kde $\lambda(v) = i$.*

3.3.1 Porovnání small progress measure vektorů

Small progress measure vektory porovnáváme lexikograficky. Operátor porovnání pracuje se dvěma vektory a přirozeným číslem n. Porovnání se provádí od nultého prvku až po n-tý prvek. Operátory porovnání značíme $\leq_n, \geq_n, >_n, <_n, =_n$. Označení $\rho(v_1) =_1 \rho(v_2)$ znamená, že nultý prvek vektoru $\rho(v_1)$ se má rovnat nultému prvku $\rho(v_2)$ a zároveň se rovnají prvky vektorů na první pozici.

Progress measure vektor může nabývat hodnoty vyjadřující nekonečno a značíme jej jako T.

Definice 14 *Pro všechny paritní hry G, existuje small progress measure vektor T takový, že $T >_i \rho(v)$ pro všechny $v \in V$ a $0 \leq i \leq d$.*

3.3.2 Inkrementace hodnot small progress measure vektorů

V průběhu výpočtu je třeba progress measure vektory upravovat. Vektor se upravuje tak, že se na určité pozici inkrementujeme o 1. Inkrementace na dané pozici se značí operátorem $+_i$, kde i určuje pozici ve vektoru. Přesáhne-li při inkrementaci hodnota vektoru na i-té pozici maximum dané i-tou pozicí mezního vektoru, dojde na této pozici k přetečení. Dojde-li k přetečení, daná pozice se vynuluje a inkrementuje se předchozí lichá pozice. Pokud inkrementace způsobí, že inkrementovaný vektor přesáhne mezní measure vektor, nastavíme jeho hodnotu na T.

3.3.3 Algoritmus SPM

Algoritmus SPM pracuje v cyklu, dokud probíhají změny progress measure vektorů. Hlavní část algoritmu je zobrazena na Výpisu 3. Funkce `init` nastaví všem vrcholům v počáteční vektor obsahující na všech pozicích nuly.

Po inicializaci jsou vrcholy procházeny v cyklu, dokud se nějaký vektor mění. Pro každý vrchol se kontroluje, zda splňují 4 podmínky stability. Tyto podmínky jsou:

1. Pokud $v \in V_0$ a $\lambda(v)$ je sudá, tak $\rho(v) \geq_{\lambda(v)} \rho(w)$ pro nějaké $(v, w) \in E$

2. Pokud $v \in V_0$ a $\lambda(v)$ je lichá, tak $\rho(v) >_{\lambda(v)} \rho(w)$ pro nějaké $(v, w) \in E$

3. Pokud $v \in V_1$ a $\lambda(v)$ je sudá, tak $\rho(v) \geq_{\lambda(v)} \rho(w)$ pro všechny $(v, w) \in E$

4. Pokud $v \in V_1$ a $\lambda(v)$ je lichá, tak $\rho(v) >_{\lambda(v)} \rho(w)$ pro všechny $(v, w) \in E$

Kontrolu uvedených podmínek zajišťuje funkce `lift`. Ta pro zkoumaný vrchol porovnává jeho vektor s vektorem následníka a příslušně jej upraví. Pokud vrchol $v \in V_0$, tak je z porovnání vybrán nejmenší measure vektor spočítaný vzhledem ke všem jeho následníkům a je dosazen jako nový vektor vrcholu. Když $v \in V_1$ je vybrán maximální vektor.

```
foreach v
   init ρ(v)
do
   foreach v
      if v ∈ V0
         ρ(v) = min{lift(v, w)|(v, w) ∈ E}
      else:
         ρ(v) = max{lift(v, w)|(v, w) ∈ E}
while change
```

Výpis 3: Hlavní část SPM algoritmu

Pokud v iteraci neproběhla žádná změna vektorů, algoritmus končí. Všechny vrcholy, jejichž progress measure vektor je nastaven na T tvoří výherní region hráče 1. Zbylé vrcholy, jejichž $\rho(v) \neq T$ tvoří výherní region hráče 0. Porovnáním progras measure vektorů následníků vrcholů hráče 0 získáme jeho výherní strategie. Optimální výherní strategie hráče 0 tvoří hrana do následujícího vrcholu s nejnižší hodnotou progress measure vektoru.

Z řešení nezískáme optimální strategie hráče 1. Pro zjištění strategií hráče 1 musíme vytvořit a vyřešit duální hru tak, že změníme vlastníka všech vrcholů a ke všem prioritám přičteme 1. Výherní strategie hráče 0 v duální hře pak odpovídají strategiím hráče 1 v původní hře.

Výsledná složitost algoritmu je $O(d \cdot m \cdot (\frac{n}{\lfloor d/2 \rfloor})^{\lfloor d/2 \rfloor})$, kde n je počet vrcholů, m je počet hran a d je maximální priorita ve hře.

3.3.4 Funkce lift

Na Výpisu 4 je uveden pseudokód funkce `lift` vracející measure vektor splňující výše uvedené podmínky. Funkce vrací vektor, který není menší než $\rho(v)$ a zároveň je větší nebo stejný jako $\rho(w)$.

Nejprve proběhne porovnání vstupních vektorů až do pozice dané $\lambda(v)$. Pokud je $\rho(v)$ větší než $\rho(w)$ nemusíme vektor $\rho(v)$ upravovat a vrátíme jej. Pokud je $\rho(v)$ menší, funkce `marge` vytvoří nový vektor $\rho'(v)$. Vektor $\rho'(v)$ obsahuje až do pozice $\lambda(v)$ shodné hodnoty jako $\rho(v)$.

Zbylé pozice jsou nastaveny na nulu. V případě, že je $\lambda(v)$ lichá inkrementujeme vektor $\rho'(v)+_{\lambda(v)}$ 1. Upravený vektor funkce navrací jako potencionálně nový progress measure vrcholu.

```
function lift
if ρ(v) ≤_λ(v) ρ(w)
   ρ'(v) = marge(ρ(v), ρ(w), λ(v))
   if λ(v)%2 != 0
      ρ'(v) = ρ'(v) +_λ(v) 1
   return ρ'(v)
else:
   return ρ(v)
```

Výpis 4: Zlepšení measure vektoru

Příklad 5

Pro potřeby SPM algoritmu převedeme hru na ekvivalentní, s minimální výherní podmínkou. Nový graf je na Obrázku 9. Po vyřešení převádíme hru zpět na hru s maximální výherní podmínkou.

Na začátku funkce `init` vytvoří pro každý vrchol progress measure vektor nastavený na $\overline{0}$. Zároveň určíme mezní vektor jako $\overline{1}$, protože každá priorita se ve hře objevuje jednou. Z mezního vektoru určíme vektor reprezentující nekonečno $T = \overline{2}$, který je na každé pozici o 1 větší než mezní vektor.

V hlavním cyklu programu porovnáváme a případně upravujeme progress measure vektory vrcholů. V první iteraci je třeba upravit vektory vrcholů v_0, v_1, v_2 a v_3. Vrchol v_0 patří hráči 1 a má lichou prioritu, a tím se na něj vztahuje podmínka č. 4. Jeho upravený vektor musí být ostře větší než vektory vrcholu v_2 a jeho vlastní. Protože jsou všechny vektory prozatím nulové, funkce `lift` vytvoří dva stejné potencionální zlepšující vektory. Oba tyto vektory jsou inkrementovány na sedmé pozici. V případě, že by se potencionální vrcholy nerovnaly, vybrali bychom z nich ten s vyšší hodnotou protože v_0 má lichou prioritu. Stejný postup aplikujeme na zbylé vrcholy. Jakmile jsou spočítány a vybrány nové vektory všech vrcholů, přepíšeme stávající vektory, pokud se od nových lišily. Pokud se nějaký z vektorů přepsal pokračuje funkce další iterací hlavního cyklu. Po první iteraci vypadají measure vektory vrcholů následovně:

1. $\rho(v_0) = \{0, 0, 0, 0, 0, 0, 0, 1\}$
2. $\rho(v_1) = \{0, 1, 0, 0, 0, 0, 0, 0\}$
3. $\rho(v_2) = \{0, 0, 0, 0, 0, 1, 0, 0\}$
4. $\rho(v_3) = \{0, 0, 0, 0, 0, 0, 0, 0\}$
5. $\rho(v_4) = \{0, 0, 0, 0, 0, 0, 0, 0\}$
6. $\rho(v_5) = \{0, 0, 0, 0, 1, 0, 0, 0\}$

Změna vektoru proběhla u 4 vrcholů. Je nutné pokračovat další iterací. Vektory po druhé iteraci:

1. $\rho(v_0) = \{0, 0, 0, 0, 0, 1, 0, 1\}$
2. $\rho(v_1) = \{0, 1, 0, 0, 0, 0, 0, 0\}$
3. $\rho(v_2) = \{0, 0, 0, 0, 0, 1, 0, 0\}$
4. $\rho(v_3) = \{0, 0, 0, 0, 0, 0, 0, 0\}$
5. $\rho(v_4) = \{0, 0, 0, 0, 0, 0, 0, 0\}$
6. $\rho(v_5) = \{0, 1, 0, 0, 1, 0, 0, 0\}$

V druhé iteraci proběhla změna u vrcholu v_0. Postupné změny probíhají až do osmé iterace, kdy už se žádný progress measure vektor nemění. Výsledný stav vektorů je:

1. $\rho(v_0) = \{T\}$
2. $\rho(v_1) = \{T\}$
3. $\rho(v_2) = \{0, 0, 0, 0, 0, 1, 0, 0\}$
4. $\rho(v_3) = \{0, 0, 0, 0, 0, 0, 0, 0\}$
5. $\rho(v_4) = \{0, 0, 0, 0, 0, 0, 0, 0\}$
6. $\rho(v_5) = \{T\}$

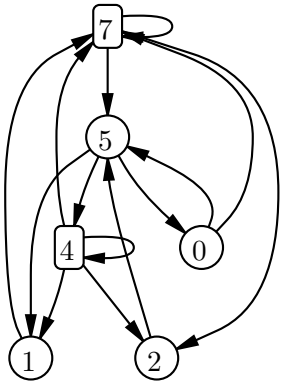
Všechny vrcholy jejichž progress measure vektor nabývá hodnoty T jsou zařazeny do výherního regionu hráče 1. Zbylé vrcholy patří do výherního regionu hráče 0. Výherní regiony hráčů jsou $W_0 = \{v_2, v_3, v_4\}$ a $W_1 = \{v_0, v_1, v_5\}$. Výherní strategie hráče 0 bychom extrahovali z řešení porovnáním vektorů následníků jeho vrcholů. Vybráním následníka s nejmenším progress measure vektorem získáme optimální strategii z daného vrcholu.

■

3.4 Strategy Improvement

Obecně algoritmus strategy improvement na počátku vychází z nějaké (nejčastěji náhodné) počáteční strategie přiřazené hráčům. Tyto strategie jsou ohodnocovány a iterativně zlepšovány. Průběh algoritmu končí v okamžiku, kdy pro žádný vrchol neexistuje strategie, která by zlepšila jeho ohodnocení. Jakmile jsou strategie ustáleny, vznikne z každého vrcholu hry jednoznačná cesta grafem. Ověřením, zda se na vzniklé cestě vyskytuje sudý nebo lichý cyklus, určíme zařazení vrcholu do výherního regionu jednoho z hráčů.

V popisu se zaměříme na algoritmus Discreate Strategy Improvement, který představili Jens Vöge a Marcin Jurdziński.[9] Tento algoritmus všem vrcholům přiřadí strukturu nazvanou herní



Obrázek 9: Upravená hra s minimální výherní podmínkou pro SPM

profil. Herní profil určuje výhodnost vrcholu vzhledem k hráči 0 a je určující pro výběr zlepšující strategie. Algoritmus v každé iteraci spočítá herní profily všech vrcholů a porovnáním vybere lepší strategii pro hráče 0, pokud existuje. Jakmile pro žádný vrchol neexistuje zlepšující strategie, výpočet končí.

Na počátku algoritmus přiřadí vrcholům hráče 0 náhodnou počáteční strategii a započne hlavní cyklus funkce. Zafixováním strategie hráče 0 vznikla v principu hra jednoho hráče, kdy všechny volby ve hře závisí na rozhodnutí hráče 1. Hráč 1 zvolí optimální proti strategii. Sloučením strategií obou hráčů vznikla jednoznačná hra. Z každého vrcholu vede pouze jedna cesta končící cyklem. Valuací této cesty se každému vrcholu přiřadí herní profil. Pro každý vrchol hráče 0 proběhne porovnání herních profilů jeho následníků. Z výsledku porovnání se vybere následník s lepším herním profilem, jako potencionální nová strategie. Jestliže je vybraná strategie jiná než aktuální, přiřadí se danému vrcholu. Proběhla-li změna strategie nějakého vrcholu, zafixují se nové strategie hráče 0 a výpočet se opakuje v další iteraci. Výpočet končí v případě, že u žádného vrcholu neproběhla změna strategie. Výsledné strategie jsou optimální pro oba hráče a herní profily obsahují informaci o zařazení do výherních regionů.

3.4.1 Herní profil

Herní profil je uspořádaná trojice (u_π, P_π, e_π), kdy u_π je hodnota partie, P_π je primární hodnota cesty a e_π je sekundární hodnota cesty. Mějme partii $\pi = (v_1, v_2, \ldots, v_l)$, která je ukončena tahem do již navštíveného vrcholu, tj. obsahuje cyklus. Hodnota partie $u_\pi = \lambda(u_k)$, kde u_k je vrchol s nejvyšší prioritou v cyklu cesty π. Dále mějme $\texttt{Prefix}(\pi) = \{v_1, v_2, \ldots, v_{k-1}\}$, tedy množinu vrcholů vyskytujících se před dosažením v_k. Primární hodnota cesty P_π je množina vrcholů z $\texttt{Prefix}(\pi)$, kde $\lambda(v) > \lambda(u_\pi)$. Sekundární hodnota cesty je délka cesty z počátečního vrcholu do u_π, tedy $e_\pi = |\texttt{Prefix}(\pi)|$.

Všechny herní profily spadají do množiny $\mathcal{D}$, která je definována jako:

$$\mathcal{D} = \{(u, P, e) \in V \times 2^V \times \{0, \ldots, |V| - 1 | \forall v \in P : u < v \wedge |p| \leq e\}$$

Herní profil určuje zařazení vrcholu do jednoho z výherních regionů W. Mějme množiny vrcholů $V_+ = \{v \in V | \lambda(v)\%2 = 0\}$ a $V_- = \{v \in V | \lambda(v)\%2 = 1\}$. Vrchol spadá do W_0, pokud má přiřazen herní profil (u, P, e) a $u \in V_+$. Pokud $u \in V_-$, tak vrchol připadne do W_1.

3.4.2 Uspořádání herních profilů

Pro porovnávání herních profilů zavedeme relevantní uspořádání vrcholů podle jejich priorit. Relevantní uspořádání setřídí vrcholy podle vhodnosti priorit pro hráče 0. Pro hráče 0 má vysoká lichá priorita nejnižší hodnotu a sudé priority nejvyšší. Relevantní uspořádání pak definujeme jako:

$$u \prec v \iff (u < v \wedge v \in V_+) \vee (v < u \wedge u \in V_-).$$

Rozšířením uspořádání na množiny mějme množiny P a Q. Nejrelevantnější vrchol jejich symetrického rozdílu $P \triangle Q$ určuje, zda $P \prec Q$. Mějme nejrelevantnější vrchol $v \in P \triangle Q$, pokud $v \in P \wedge v \in V_-$, pak $P \prec Q$. Pokud $v \in P \wedge v \in V_+$, pak $Q \prec P$.

Na množině herních profilů můžeme definovat relevantní uspořádání $\prec$. Mějme herní profily $(u, P, e), (v, Q, f)$, pak relevantní uspořádání definujeme jako:

$$(u, P, e) \prec (v, Q, f) \iff \begin{cases} u \prec v \\ \vee (u = v \wedge P \prec Q) \\ \vee (u = v \wedge P = Q \wedge v \in V_- \wedge e < f) \\ \vee (u = v \wedge P = Q \wedge v \in V_+ \wedge e > f) \end{cases}$$

Pokud se hodnoty partie a primární hodnoty cest dvou herních profilů rovnají, tak je rozhodujícím faktorem sekundární hodnota cesty. Pokud vrchol $v \in V_+$, je pro hráče 0 žádoucí dostat se do tohoto vrcholu co nejkratší cestou. Pokud $v \in V_-$ hráč 0 se snaží oddálit navštívení v a vybírá delší cestu.

3.4.3 Valuace

Mějme počáteční vrchol v, strategii f_0 a f_1. Společně tvoří unikátní partii π_{v,f_0,f_1}. Valuace je funkce $\varphi : V \to \mathcal{D}$, která každému vrcholu přiřadí herní profil vzhledem k π_{v,f_0,f_1}. V případě potřeby můžeme jednotlivé části herního profilu vrcholu referovat jako $\varphi_0(v) = u$, $\varphi_1(v) = P$, $\varphi_2(v) = e$.

3.4.4 Optimální valuace a zlepšení strategie

Valuace $\varphi(u) \prec \varphi(v)$ je lepší pro hráče 0 právě tehdy, když herní profil vrcholu v je vzhledem k relevantnímu uspořádání lepší než herní profil vrcholu u.

Valuaci nazýváme optimální pro hráče 0, pokud neexistuje žádný jiný následník vrcholu, který by měl lepší valuaci. Valuace φ je optimální pro hráče 0 a vrcholy $x, y \in V$ s hranou xEy, když $\forall z \in V : xEz \Rightarrow \varphi(z) \prec \varphi(y)$.

Výběr lepší strategie spočívá ve vybrání následníka s maximální valuací φ. Mějme valuace φ a φ', valuaci φ' nazveme zlepšením strategie hráče 0 pokud:

$$\forall x \in V_0 \; \exists y \in V : xEy \wedge \varphi(x) \prec \varphi(y) \wedge \forall z \in V : xEz \Rightarrow \varphi(z) \prec \varphi(y).$$

3.4.5 Výherní strategie a složitost

Jakmile se v poslední iteraci nezmění žádná valuace výpočet algoritmu končí. Na konci výpočtu máme k dispozici optimální valuace všech vrcholů, a tím i jejich strategie. Výherní regiony určíme jako $W_0 = \{v|\varphi_0(v) \in V_+\}$ a $W_1 = \{v|\varphi_0(v) \in V_-\}$. Výherní strategie odpovídají získaným optimálním strategiím. Pro hráče 0 jsou to hrany, které vedou do vrcholů s maximální valuací. Výherní strategie hráče 1 jsou hrany, které vedou do vrcholů s minimální valuací.

Při určování výpočetní složitosti zohledňujeme dva problémy. Prvním je určení složitosti jedné iterace algoritmu. Druhým problémem je určení počtu iteracích nutných k získání optimálních strategií. Většina algoritmů má složitost zhruba $O((n/d)^d)$, kde d je počet různých priorit ve hře. Nicméně autoři prozatím nebyli schopni určit složitost přesněji než 2^n. Prozatím nejlepší horní hranice je stále exponenciální, ale zatím nebyly nalezeny příklady her, které by potřebovaly více jak lineární počet iterací.

Příklad 6

Pro ukázku funkce algoritmu Strategy Improvement použijeme hru na Obrázku 8. Nejprve vybereme počáteční strategii hráče 0. Vybraná strategie se pro hráče 0 zafixuje. Následuje vstup do hlavního cyklu algoritmu a nastavení strategií hráče 1 proti zafixované strategii hráče 0. Tím vznikne jediný možný postup hrou, kdy strategie obou hráčů jsou zafixovány. Tyto strategie vytvoří nový graf hry, který slouží pro valuaci. Nový graf je vyobrazen na Obrázku 10. Zvolené strategie hráče 0 jsou: $v_1 \to v_0$, $v_2 \to v_5$, $v_3 \to v_2$, $v_4 \to v_2$. Spočítaná strategie hráče 1 je: $v_0 \to v_2$, $v_5 \to v_1$.

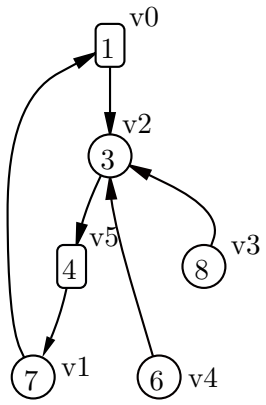
Na vzniklé podhře provedeme valuaci hry. Každému vrcholu se spočítá a přiřadí jeho herní profil. Herní profily vrcholů jsou:

---

v_0 - (1,{},3)
v_1 - (1,{},0)
v_2 - (1,{},2)
v_3 - (1,{3},3)
v_4 - (1,{},3)
v_5 - (1,{},1)

Následně se pro vrcholy hráče 0 provede srovnání valuací. Pro každý vrchol hráče 0 se porovnají herní profily jeho následníků a vybere se ten nejlepší. Novou strategií z vrcholu se stává hrana do následníka s nejlepší valuací.

U vrcholů v_1 a v_4 není třeba kontrolovat valuace, protože mají pouze jednu odchozí hranu. Pro vrchol v_2 porovnáme herní profily vrcholů v_1, v_3 a v_5. Z porovnání vychází jako nejlepší



Obrázek 10: Hra v první iteraci s zafixovanými strategiemi obou hráčů

herní profil vrcholu v_3. U vrcholu v_3 zkoumáme herní profily vrcholů v_0 a v_2, kde nejlepší je herní profil v_0. Hráči 0 se tím změnila strategie na dvou vrcholech a to: $v_2 \to v_3$, $v_3 \to v_0$. Jelikož nastala změna strategií postup se opakuje.

Určením nové strategie hráče 1 vznikl graf hry na Obrázku 11. Valuací grafu jsou vrcholům přiřazeny tyto valuace:

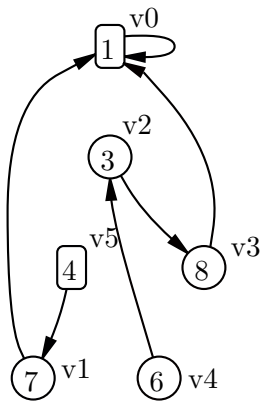
---

v_0 `- (0,{},0)`
v_1 `- (0,{1},1)`
v_2 `- (0,{2,3},2)`
v_3 `- (0,{3},1)`
v_4 `- (0,{2,3,4},3)`
v_5 `- (0,{1,5},2)`

Valuace jsou porovnány pro vrcholy v_2 a v_3. Pro vrchol v_2 je nejvýhodnější herní profil vrcholu v_3 a pro vrchol v_3 je to herní profil vrcholu v_2. Srovnáním a vybráním nejlepšího následníka se změnila strategie pouze vrcholu $v_2 \to v_3$.

V další iteraci se provede výpočet na grafu se změněnou strategií vrcholu v_3. Určením nové valuace vrcholů a jejich porovnáním již nenastane žádná změna strategie. Ze získaných valuací získáme výherní regiony. Pokud nejrelevantnější vrchol uvedený v herním profilu vrcholu je sudý, tak vrchol spadá do výherního regionu hráče 0. Pokud je lichý, tak do výherního regionu hráče 1. Výherní regiony jsou $W_0 = \{v_2, v_3, v_4\}$ a $W_1 = \{v_0, v_1, v_5\}$ a výherní strategie $v_0 \to v_0$, $v_2 \to v_3$, $v_3 \to v_2$, $v_4 \to v_2$, $v_5 \to v_1$.

■



Obrázek 11: Hra v druhé iteraci s zafixovanými strategiemi obou hráčů

4 PGSolver

PGSolver je softwarový nástroj určený k řešení paritních her. Jeho autoři Oliver Friedmann a Martin Lange jej vytvořili v rámci institutu informatiky na Mnichovské univerzitě v Německu. Tento nástroj ve zkratce představíme a shrneme jeho možnosti.

PGSolver je open source aplikace pro linuxové prostředí určená k řešení paritních her. Zdrojové kódy jsou k dispozici ke stažení na GitHubu `https://github.com/tcsprojects/pgsolver`.[11] Aplikace je implementována v jazyce OCaml. Pro zprovoznění aplikace je třeba stáhnou zdrojové soubory, překladač pro jazyk OCaml a nástroj pro kompilaci GNU make (autoři doporučují alespoň verzi 3.09.2 pro OCaml překladač a 3.81 pro GNU make). Pro některé algoritmy je třeba doinstalovat program pro řešení SAT problému. Součástí zdrojových kódů je dokumentace, která obsahuje stručné shrnutí paritních her, seznam obsažených algoritmů, krátký popis funkčnosti a stručnou vývojářskou příručkou.

4.1 Funkce PGSolveru

PGSolver se spouští voláním spustitelného souboru s parametry z terminálu. Obecná struktura pro spuštění je `pgsolver [options] [infile]`. Seznam možných nastavení a parametrů je obsažen v dokumentaci. Nejdůležitějšími parametry jsou:

- `-solver` slouží pro výběr algoritmu. Za solver se dosadí název algoritmu viz. Tabulka 1.
- `-v` určuje množství výpisů generovaných aplikací na výstup terminálu. Lze nastavit v rozmezí 0-3.

Spuštění aplikace se správnými parametry může vypadat následovně.

```
./pgsolver -global recursive pg.gm
```

Příkaz volá aplikaci PGSolver s parametry `-global recursive` a `pg.gm`. První parametr označuje použití rekurzivního algoritmu, druhý parametr určuje soubor, případně cestu k souboru, ve kterém je uložena paritní hra. Po úspěšném vyřešení hry aplikace vypíše řešení hry. Řešení obsahuje čas potřebný k vyřešení, výherní regiony a výherní strategie. Následující text prezentuje výpis řešení v terminálu.

```
PGSolver Collection Ver. 3.4: Parity Game Solver
Authors: Oliver Friedmann (University of Munich) and
Martin Lange (University of Kassel), 2008-2015
http://www.tcs.ifi.lmu.de/pgsolver
Parsing ............................... 0.00 sec
Chosen solver ‘recursive’ ............. 0.00 sec
Player 0 wins from nodes:
```

Algoritmus	Název parametru
Strategy improvement (Jurdziński, Vöge)	-stratimprove nebo -si
Small progress measure (Jurdziński)	-smallprog nebo -sp
Resursive algorithm (Zielonka)	-recursive nebo -re
Dominion decomposition (Jurdziński, Paterson, Zwick)	-dominiondec nebo -dd

Tabulka 1: Tabulka vybraných algoritmů obsažených v aplikaci PGSolver

```
  {0, 3, 5, 9}
with strategy
  [0->9,3->3,5->5,9->3]

Player 1 wins from nodes:
  {1, 2, 4, 6, 7, 8}
with strategy
  [1->8,2->7,4->7,6->7,7->4,8->2]
```

4.1.1 Generátor náhodných her

Součástí instalace aplikace PGSolver je generátor náhodných her. Příkaz pro spuštění generátoru vypadá následovně:

```
./randomgame # # # #
```

Význam jednotlivých znaků # zleva je - počet vrcholů, maximální priorita vrcholu, minimální počet odchozích hran z vrcholu, maximální počet hran z vrcholu. Příkaz pro vygenerování a vyřešení náhodné hry rekurzivním algoritmem s 10 vrcholy, maximální prioritou 10 a 2-4 odchozími hranami vypadá následovně:

```
./randomgame 10 10 2 4|./pgsolver -global recursive
```

Vygenerovanou hru lze vypsat do terminálu nebo exportovat do souboru.

4.1.2 Benchmark

Další součástí instalace PGSolveru je benchmark aplikace. V benchmarku lze porovnávat algoritmy mezi sebou a nechat si vypsat výsledné časy řešení. Pro spuštění benchmarku slouží příkaz:

```
./benchmark [options]
```

`Options` může obsahovat například tyto parametry:

- `filename` název souboru obsahující paritní hru

- `-solver` slouží pro výběr algoritmů. Za solver se dosadí název algoritmu viz. Tabulka 1 včetně jeho případných parametrů. Pokud nejsou zadávány parametry, je nutno zadat prázdné apostrofy ". Zadaných algoritmů může být více.
- `-times` počet opakování testu.

Řešení hry algoritmy zadanými v parametrech se provede 10x, pokud parametr `times` nenastavil jiný počet opakování. U každého řešení se zaznamenává doba průběhu. Výpis výsledků obsahuje nejlepší, nejhorší a průměrný čas řešení. Aplikace neumožňuje jiné měření výkonu algoritmu než časové porovnání. Výstup testování je textově do terminálu a obsahuje informace o času trvání jednotlivých průběhů testování. Po vypsání všech průběhů zobrazí souhrnnou tabulku.

Vzorový příkaz `./randomgame 1000 1000 2 10 | ./benchmark -re" -times 5` spustí benchmark na náhodné hře o 1000 vrcholech. Testovaný algoritmus je rekurzivní a testování se provede 5x. Výpis výsledku benchmarku do terminálu vypadá následovně:

```
Benchmarking recursive...
Game # 0, Iteration #1...0.01 sec
Game # 0, Iteration #2...0.01 sec
Game # 0, Iteration #3...0.01 sec
Game # 0, Iteration #4...0.01 sec
Game # 0, Iteration #5...0.01 sec
Finished. Best: 0.01 sec  Avg: 0.01 sec
Worst: 0.01 sec

+-------------------------------------------------+
| Benchmark Statistics                            |
+-------------------------------------------------+
| Solver      |     Best |  Average |    Worst    |
+-------------------------------------------------+
| recursive   | 0.01 sec | 0.01 sec | 0.01 sec    |
+-------------------------------------------------+
```

4.1.3 Shrnutí

Aplikace PGSolver po letech vývoje nabízí několik různých algoritmů pro řešení paritních her a umožňuje uživateli vyzkoušet řešení svých nebo náhodných her. Společně s implementovanými algoritmy obsahuje optimalizace, které mají zrychlit výpočet řešení her. Součástí instalace je generátor náhodných her, použitelný jako přímý vstup do aplikace nebo jako zdroj her do jiných aplikací. Další součástí instalace je benchmark umožňující časové srovnávání algoritmů.

Zprovoznění a práce s aplikací je mírně problematická z důvodu neaktuální dokumentace. Stává se, že uvedené parametry nebo příklady způsobují chybová hlášení. Nápověda obsažená přímo v programu ale dokáže většinu problémů se zadáním vyřešit.

4.2 Formát paritní hry pro uložení

PGSolver používá pro reprezentaci paritních her speciální formát. Zápis paritní hry se skládá z nepovinné hlavičky a seznamu vrcholů. Specifikace vrcholu obsahuje identifikátor, prioritu, vlastníka vrcholu, seznam následníků a nepovinný název vrcholu. Tento format můžeme vyjádřit v EBNF.

```
<parity_game> ::= [parity <identifier> ;] <node_spec>+
<node_spec>   ::= <identifier> <priority> <owner> <successors> [<name>] ;
<identifer>   ::= N
<priority>    ::= N
<owner>       ::= 0 | 1
<successors>  ::= <identifier> (, <identifier>)*
<name>        ::="text neobsahující uvozovky"
```

Pokud je v zápise hry obsažena hlavička s klíčovým slovem `parity`, následující identifikátor značí nejvyšší identifikátor vrcholu. Všechny další řádky po hlavičce obsahují definice vrcholů. Definice vrcholu `node_spec` musí mezi jednotlivými složkami obsahovat mezeru a je ukončena středníkem na konci řádku. První složkou definice vrcholu je `identifier` a je to přirozené číslo jednoznačně označující vrchol. Zbylé složky jsou: `priority` - přirozené číslo přiřazující vrcholu prioritu, `owner` - vlastnictví vrcholu hráčem 0 nebo 1, `successors` - identifikátory následníků vrcholu odděleny čárkami, `name` - nepovinný slovní název vrcholu.

Příklad správně vytvořeného zápisu hry může vypadat následovně:

```
parity 6;
1 3 0 1,3,4 "Europe";
2 6 1 4,2 "Africa";
3 5 1 0 "Antarctica";
4 8 1 2,4,3 "America";
5 6 0 4,2 "Australia";
6 7 0 3,1,0,4 "Asia";
```

5 Softwarový nástroj řešící paritní hry

Jedním z cílů práce je vytvoření aplikace, která by dokázala řešit paritní hry algoritmy představenými v Kapitole 3. Vzniklá aplikace dokáže paritní hry řešit a také provádět testování algoritmů. Testování neprobíhá pouze za pomoci měření časové náročnosti jako u PGSolveru, ale přidává i další možnosti. Aplikace má přehledné GUI pro zjednodušení ovládání, zobrazování informací a výsledků z řešení.

5.1 Popis vlastního navrhovaného programu pro řešení paritních her

Aplikace je vytvořena s ohledem na to, aby byla jednoduchá a přehledná na ovládání. Proto je tvořena jako aplikace s okny, oproti například PGSolveru, který je terminálovou aplikací. V oknech aplikace jsou přístupné všechny ovládací prvky potřebné pro řešení her nebo testování algoritmů. Aplikace je pro přehlednost rozdělena do dvou částí, mezi kterými se lze přepínat. Jedna část aplikace je zaměřena na řešení her a je nazvána `Solver`. Zatímco druhá část, pojmenovaná `Benchmark`, je zaměřena na testování algoritmů.

Obě části aplikace mají společné určité prvky. Hlavním společným prvkem jsou algoritmy pro řešení paritních her. Dalším společným prvkem jsou možnosti načítání her do aplikace. Možnosti, jak nahrát hru jsou celkem čtyři.

1. Nahrání paritní hry ze souboru, který obsahuje textový předpis hry ve formátu uvedeném v části 4.2.
2. Vygenerování náhodné hry podle zadaných parametrů.
3. Zadání požadované hry ve formuláři.
4. Vygenerování série náhodných her podle zadaných parametrů.

Při nahrávání hry ze souboru je nutné přesně dodržet daný formát. V části `Solver` lze nahrát vždy pouze jeden soubor hry. Část `Benchmark` dovoluje nahrát více souborů najednou. Generování náhodné hry je shodné pro obě části aplikace. Je zapotřebí zadat údaje o počtu vrcholů, maximální prioritě a minimu a maximu možných odchozích hran z vrcholu. Pro ruční zadávání her je připravena tabulka, do které se na řádky zadají informace o vrcholech. Vstup je ověřen a vytvoří zadanou hru. Čtvrtou možnost nahrání hry obsahuje pouze část `Benchmark` a generuje hry s počtem vrcholů v zadaných mezích.

Část `Solver` umožňuje vyexportovat nahrané nebo vytvořené hry. Tato část aplikace je také schopna načtené hry vizualizovat. Vizualizace probíhá buď v textovém, nebo v grafickém formátu. V grafickém zobrazení aplikace vykreslí zadaný graf a v případě, že je hra vyřešena označí výherní regiony v grafu. Zobrazování grafické podoby grafu je omezeno pouze do určité velikosti hry. Vykreslování je omezeno z důvodu náročnosti vykreslování velkého množství vrcholů a hran.

Primární funkcí části `Solver` je řešení her. Po vyřešení hry se do textového výstupu vypíší všechny informace získané při řešení. Do textového výstupu se vypisují stavové informace a následně řešení hry. K dispozici je možnost zapnutí detailního výpisu, který vypisuje informace o průběhu řešení.

`Benchmark` část je zaměřená na srovnávání algoritmů mezi sebou. Ze seznamu algoritmů je možné vybrání jednoho až všech a provést testování. Pro testování je možné zvolit kolik opakování každého testu se má provést.

Pro zobrazení výsledků testování je přichystána tabulka a graf pro zobrazení naměřených hodnot. Při testování se sledují hodnoty o tom, jak dlouho trval výpočet, kolik bylo zpracováno vrcholů a kolik bylo provedeno základních operací, kdy základní operace je například rekurzivní volání nebo iterace hlavního cyklu. Počet zpracovaných vrcholů pak označuje počet čtení informací z instancí vrcholů v průběhu výpočtu. Mezi zobrazením těchto tří sledovaných výsledků lze přepínat. Získané výsledky je možno exportovat do csv souboru pro další zpracování.

5.2 Zvolená platforma a programovací jazyk

Pro implementaci aplikace byla zvolena platforma Windows. Zvolena byla jak z důvodu, že Windows je stále nejrozšířenější platformou na světě, tak z důvodu, že pro linuxové prostředí již existuje podobný nástroj, zmiňovaný PGSolver. Aplikace je díky tomu použitelná pro obrovskou skupinu potencionálních uživatelů.

Jako programovací jazyk pro implementaci byl zvolen C#.[12] Jedná se o objektově orientovaný jazyk fungující na .NET Frameworku. Použitá verze frameworku je 4.5.2. Jedná se o jazyk z rodiny jazyků C a jeho syntaxe je tedy velice podobná jazykům C/C++ a Java. Existuje pro něj přes 4000 knihoven obsažených v .NET Frameworku zajišťující jednoduchou a rychlou práci počínaje vstupy/výstupy až například Windows Forms komponentami. Jedna z výhod je i pokročilé vývojové prostření Visual Studio a rozsáhlá dokumentace na webu MSDN.

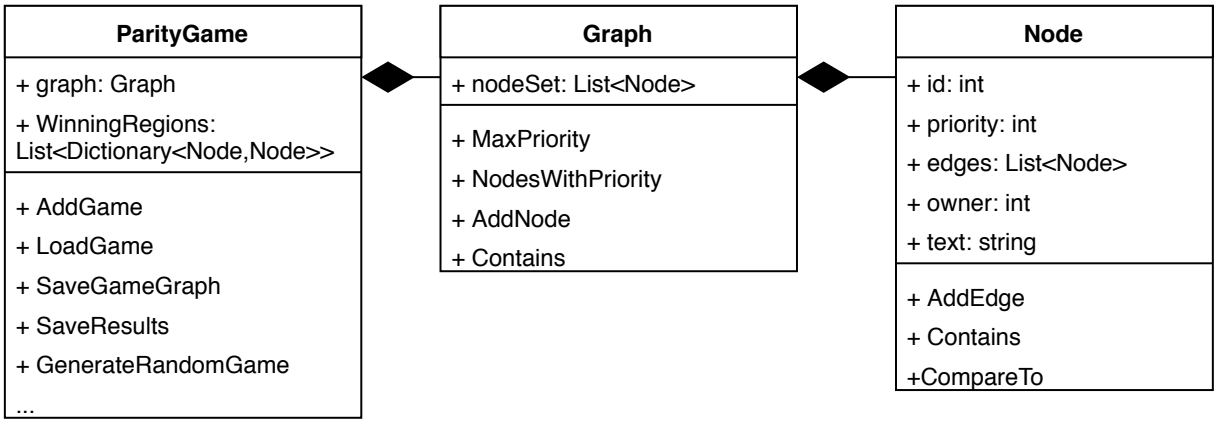
5.3 Objekty reprezentující paritní hry

Pro reprezentaci paritní hry v aplikaci vznikly tři třídy. Třída `ParityGame` reprezentuje paritní hru. Paritní hra se skládá z grafu a graf z vrcholů. Graf hry je tvořen třídou `Graph` a jednotlivé vrcholy třídou `Node`. Vztah objektů je zobrazen na Obrázku 12.

5.3.1 Třída Node

Třída `Node` je základním stavebním kamenem pro výstavbu grafové struktury. Jedná se o třídu reprezentující vrcholy grafu. Třída obsahuje své atributy a to jmenovitě:

- `int id` - integer označující daný vrchol. Identifikátor koresponduje s označením používaným v grafu paritních her tj. $v_1 \dots v_n$.
- `int owner` - integer nabývající pouze hodnot {0,1} určující vlastníka vrcholu.



Obrázek 12: Objekty reprezentující paritní hru

- `int priority` - integer určující prioritu vrcholu.
- `List<node> edges` - list všech vrcholů, které jsou následníkem daného vrcholu.
- `string text` textový popisek vrcholu.
- `double access` počítadlo přístupů k vrcholu.

Třída pro svou funkci obsahuje dvě metody. Metoda `AddEdge(Node n)` zajišťující přidání nové hrany. Metoda `Contains(int id)` zjišťuje podle zadaného ID vrcholu, zda má vrchol následníka s daným ID.

5.3.2 Třída Graph

Třída `Graph` reprezentuje instanci grafu hry. Obsahuje pouze jeden vlastní atribut `List<Node> nodeSet`. Atribut slouží pro uložení všech vrcholů daného grafu a je to list objektů typu `Node`.

Pro práci s objektem typu `Graph` uvedeme vybrané metody, zajišťující přístup k informacím o grafu a jeho vrcholech. Tyto metody jsou:

- `public void AddNode(Node node)` přidá objekt typu `Node` jako vrchol do grafu hry.
- `public Node Contains(int id)` kontroluje existenci vrcholu v grafu podle zadaného ID. Metoda vrací nalezený vrchol typu `Node`.
- `public int MaxPriority(List<int> removed)` metoda vyhledá v grafu hry nejvyšší prioritu. Její argument dodává omezující podmínku pro vyhledávání. Je to list ID vrcholů, které jsou aktuálně neaktivní.
- `public List<Node> NodesWithPriority(int priority, List<int> removed)` metoda vyhledá vrcholy se zadanou prioritou. Vyhledávání je omezeno pouze na aktuálně aktivní vrcholy.

Třída obsahuje i několik dalších pomocných metod, které nebudeme zmiňovat a příslušné konstruktory. V některých metodách je použit list ID neaktivních vrcholů. Je použit z důvodu, že u některých algoritmů je potřeba pracovat s určitými podgrafy, ale není potřeba vytvářet nové grafy složené pouze z aktivních vrcholů.

5.3.3 Třída ParityGame

Třídu `ParityGame` můžeme označit jako hlavní třídu programu, která představuje instanci paritní hry. Obsahuje atributy `graph` a `winningRegions`. Atribut `graph` vyjadřuje herní graf a atribut `winningRegions` představuje řešení hry. Atribut `winningRegions` je typu `List<Dictionary<Node, Node> >`. List obsahuje 2 slovníky (jeden pro každého hráče), kde příslušný slovník obsahuje výherní vrcholy hráče s příslušnými strategiemi.

Kromě atributů třída obsahuje obslužné metody. Pro načítání a ukládání her slouží tyto metody:

- `LoadGame` - načte graf hry ze souboru s předpisem hry.
- `SaveGameGraph` - uloží graf hry jako soubor do vybraného adresáře.
- `SaveResult` - uloží výherní regiony a strategie jako soubor do vybraného adresáře.

Další z metod, které zmíníme, je `GenerateRandomGame` a `Attractors`. První z nich zajišťuje náhodné generování grafu hry podle zadaných parametrů. Druhá metoda počítá v grafu hry atraktory k zadaným vrcholům. Atraktory využívá jak rekurzivní algoritmus, tak algoritmus hledání malých dominií. Třída navíc obsahuje množství obslužných metod, zajišťující přístup například ke stavům a další operace.

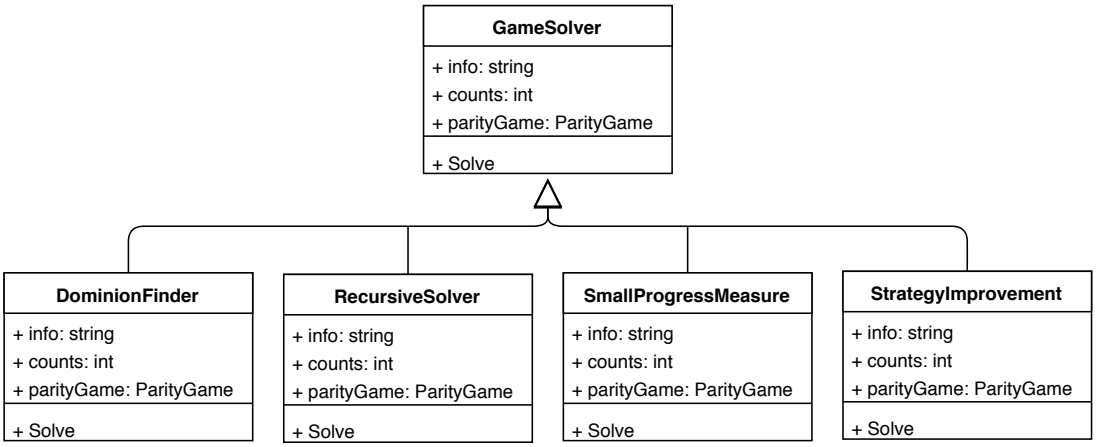
5.4 Třídy realizující algoritmy

5.4.1 Třída GameSolver

K řešení her je definována samostatná množina tříd. Třída `GameSolver` je abstraktní třída, která poskytuje základní rozhraní pro všechny třídy, implementující řešící algoritmus, které z ní dědí. Třída obsahuje atribut pro uchování paritní hry a několik atributů pro uložení informací jako například objekt pro textové zaznamenání průběhu hry.

Kromě konstruktoru obsahuje třída předpis metody `Solve`. Jejími parametry jsou paritní hra a seznam neaktivních vrcholů. Návratová hodnota metody `Solve` je řešení paritní hry ve formě výherních regionů a strategií. Navíc metoda ve výstupním parametru vrací textový průběh hry, pokud byl požadován.

Řešení pomocí abstraktní třídy, ze které ostatní třídy dědí, nám umožňuje zjednodušit a unifikovat použití všech tříd pro řešení her. Hierarchie tříd typu `GameSolver` je na Obrázku 13.



Obrázek 13: Hierarchie tříd typu GameSolver

5.4.2 Třída RecursiveSolver

Třída `RecursiveSolver` dědí z rodičovské třídy `GameSolver` a implementuje rekurzivní algoritmus. Třída obsahuje pouze jednu zděděnou metodu `Solve`, která obstarává vyřešení hry. Obsah metody koresponduje s pseudokódem uvedeným v Kapitole 3.1. Metoda nejprve kontroluje, zda není vstupní hra prázdná. Pokud je hra prázdná vrací prázdné řešení. To je ukončující podmínka rekurze. Pokud vstupní hra není prázdná, vykoná se rekurzivní volání metody `Solve`. Průběh těla této metody byl již popsán v části zabývající se rekurzivním algoritmem.

Z důvodu práce s podhrami v jednotlivých rekurzivních voláních byl zaveden vstupní parametr `Removed`. Tento parametr slouží jako seznam aktuálně neaktivních vrcholů. V průběhu řešení nahrazuje tvorbu nových podgrafů.

V průběhu řešení se sbírají informace o postupu řešení. U rekurzivního algoritmu se jedná o hloubku zanoření, maximální prioritu nebo aktuálně tvořený atraktor. Tyto informace jsou zaznamenávány v textové formě a jsou vraceny jako výstupní parametr společně s řešením.

5.4.3 Třída DominionFinder

Třída `DominionFinder` reprezentuje algoritmus hledání malých dominií. Kromě metody `Solve`, která je předepsaná rodičovskou třídou, obsahuje třída metodu `SolveRec`. Metoda `Solve` obsahuje kód nastíněný ve Výpisu 2 a metoda `SolveRec` obsahuje upravený kód rekurzivního algoritmu, využívaný v případě, že nebylo nalezeno žádné dominium.

Klíčovou částí algoritmu je vyhledání dominia v grafu hry. Tuto funkci zastává metoda `Dominium` a její průběh je zobrazen na Obrázku 14. V rámci této metody se musí vygenerovat všechny podmnožiny vrcholů grafu do dané velikosti a ověřit, zda splňují podmínky na to být dominiem. Metoda přijímá jako parametry graf hry, seznam neaktivních vrcholů a velikost hledaných dominií. Výstupem je množina vrcholů a číslo označující, zda se jedná o 0-dominium

nebo 1-dominium. K zajištění požadované funkce jsou potřeba tři pomocné metody `IsClosed`, `IsDominium` a `Combinations`.

Metoda `Combinations` slouží při hledání dominií jako enumerátor, který postupně generuje podmnožiny vrcholů grafu. Je použit v rámci cyklu, kde v každé iteraci poskytne jednu vygenerovanou podmnožinu. Na každé podmnožině je proveden test, na základě, kterého zjistíme, zda je uzavřená. Podmnožiny jsou nejprve kontrolovány metodou `IsClosed`, která provádí test uzavřenosti. Test je potřeba provést dvakrát, jednou pro každého z hráčů. Pokud množina není uzavřená, pokračuje generování podmnožin. V případě, že je množina uzavřená pro jednoho z hráčů, metoda `IsDominium` kontroluje, zda jsou vrcholy výherní pro tohoto hráče. Jsou-li všechny vrcholy výherní pro hráče, vůči kterému je množina uzavřená, tak bylo nalezeno dominium. Při nalezení dominia končí generování dalších podmnožin grafu a metoda `Dominium` vrací tuto množinu jako výherní pro jednoho z hráčů.

Bylo-li nalezeno dominium, tak je k jeho vrcholům vytvořen příslušný atraktor a rekurzivně se volá funkce `Solve` na hru s odstraněným atraktorem. Jestliže nebylo dominium nalezeno, řešení hry pokračuje voláním metody `SovleRec`.

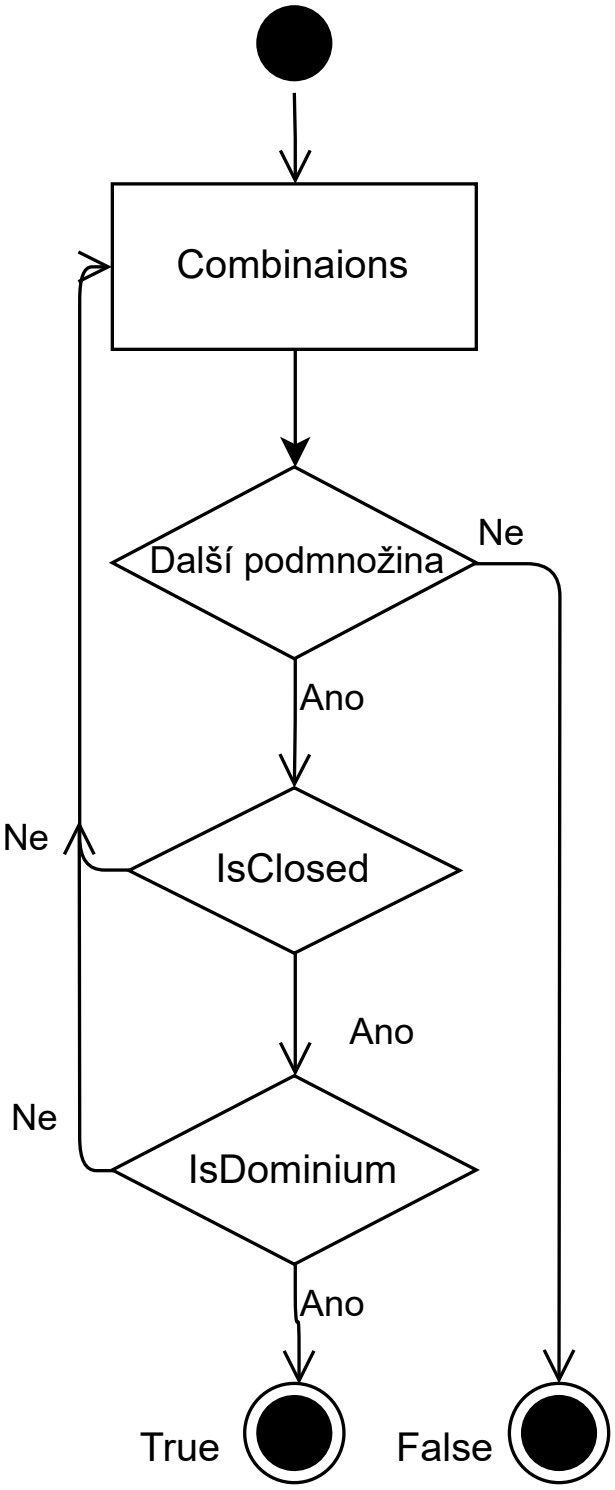
5.4.4 Třída SmallProgressMeasure

Algoritmus Small Progress Measure je reprezentován třídou `SmallProgressMeasure`. Tento algoritmus předpokládá hru ve formátu min-parity. V aplikaci předpokládáme max-parity formát. Proto potřebujeme hru před začátkem řešení a po skončení výpočtu upravit podle Definice 6. Pro převod mezi max a min parity je na začátku a konci metody `Solve` volána metoda `ChangeParity`, která převod provede.

Jakmile máme hru ve formátu min-parity, inicializujeme všechny vrcholy prázdným prograss measure vektorem. Inicializaci provádí metoda `Init`, která vytvoří slovník vektorů. Slovník obsahuje jako klíč identifikátor vrcholu pro snadný přístup a každému klíči přiřadí prázdný progress measure vektor. V C# je slovník implementován jako hash tabulka a proto je možné k uloženým vektorům přistupovat pomocí klíče se složitostí blízkou $O(1)$. Při průchodu všemi vrcholy zároveň vytváří mezní vektor - `boundVector`. Po průchodu všemi vrcholy a získání mezního vektoru, vytvoří z mezního vektoru vektor T jako proměnnou `infVector`.

Po inicializaci procházíme v cyklu všechny vrcholy grafu hry, dokud se nějakému vrcholu mění jeho vektor. Pro každý vrchol spočítáme metodou `Progress` nový progress measure vektor vůči jeho následníkům. Pokud vrchol patří hráči 0, generujeme nové progress measure vektory vůči všem jeho následníkům a metodou `Compare` mezi nimi vybíráme vektor s minimální hodnotou. Pokud je nejmenší vygenerovaný vektor větší než vektor zpracovávaného vrcholu, tak tento vektor přiřadíme zpracovávanému vrcholu jako jeho nový vektor. Pokud provedeme změnu vektoru vrcholu nastavíme příznak `change`. Jestliže vrchol patří hráči 1, z porovnávaných vektorů vybíráme ten největší.

Jestliže je nastaven příznak `change`, tak byl v dané iteraci změněn nějaký measure vektor a výpočet pokračuje další iterací. Po skončení výpočtu jsou zkontrolovány progress measure



Obrázek 14: Hledání dominia

vektory všech vrcholů a podle jejich hodnoty jsou zařazeny do výherních regionů. Vektory s hodnotou T vyhrává hráč 1 a zbývající hráč 0.

V průběhu zpracování vrcholů jsou generovány measure vektory metodou `Progress`. Metoda `Progress` vytváří nový progress measure vektor aktuálně zpracovávaného vrcholu vzhledem k jednomu z jeho následníků. Pokud je aktuální vektor vrcholu menší nebo roven vektoru následníka, tak se vytvoří nový vektor, jehož prvních $\lambda(v)$ prvků je kopií vektoru následníka a zbylé prvky jsou nastaveny na 0. Pokud má vrchol v lichou prioritu, metoda `Increment` zvýší hodnotu vektoru na pozici $\lambda(v)$ o 1.

Metoda `Increment` kontroluje vůči meznímu vektoru, zda na dané pozici nedošlo k přetečení. Pokud došlo k přetečení, rekurzivně zavolá sama sebe a snaží se inkrementovat nejbližší nižší lichou pozici. V případě, že je inkrementovaná pozice nižší než 1, dochází k přetečení mezního vektoru. Přetečení mezního vektoru ukončuje rekurzi a nastavuje vektor na T. Nedojde-li k přetečení, danou pozici inkrementuje o 1.

Vektory generované metodou `Progress` jsou porovnávány mezi sebou metodou `Compare`. Metoda porovnává dva vektory od pozice 0 až po pozici n danou vstupním parametrem. Návratové hodnoty jsou 0 v případě rovnosti vektorů, 1 pokud je první z vektorů větší a -1 pokud je menší.

5.4.5 Třída StrategyImprovement

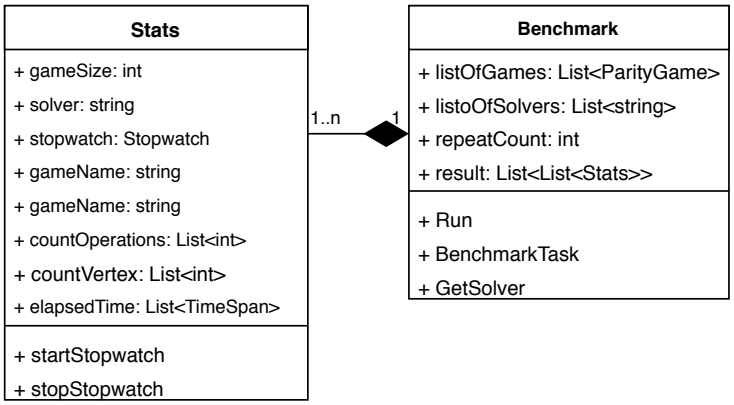
Instance třídy `StrategyImprovement` implementuje algoritmus zlepšujících strategií, který byl popsán v Kapitole 3.4. Hlavní část algoritmu je obsažena v metodě `Solve`, která obsahuje hlavní cyklus algoritmu. Pro účely tohoto algoritmu vzniklo mnoho pomocných metod a několik pomocných tříd.

Instance třídy `Valuation` uchovávají valuace jednotlivých vrcholů a instance třídy `Strategy` představují strategii vrcholu. V průběhu řešení je potřeba porovnávat jednotlivé valuace vrcholů. Porovnávání zastává statická třída `RewardOrder`. Její primární metoda `LessThan` porovnává dvě valuace podle relevantního uspořádání herních profilů zavedeného v Kapitole 3.4.2.

V samém začátku řešení je provedena metodou `RandomInit` inicializace hry. Inicializace spočívá ve vybrání náhodných počátečních strategií pro hráče 0. Za pomocí těchto strategií je vygenerován nový graf hry, kde ze všech vrcholů hráče 0 vychází pouze jedna hrana a je určena strategie hráče 1.

Generování nových grafů hry zajišťují metody `CreateSubgraph` a `CreateFullSubgraph`. První ze zmíněných metod generuje upravený graf prostřednictvím strategií pouze jednoho z hráčů. Druhá z metod generuje graf, kde každý vrchol má pouze jednu výstupní hranu, podle aktuálních strategií hráčů. Takto vytvořený graf je použit pro valuaci hry.

Získáním grafu a jednoznačných strategií obou hráčů můžeme provést valuaci hry. Valuaci grafu hry provádí metoda `Valuate`. V této metodě je potřeba vytvořit z každého vrcholu cestu grafem podle nastavených strategií. Pomocné metody postupně projdou a zaznamenají cesty ze všech vrcholů. Cesta grafem je ukončena v okamžiku, kdy při pohybu z počátečního vrcholu



Obrázek 15: Třídy Benchmark a Stats

dojdeme do vrcholu, který již byl jednou navštíven. Zaznamenané cesty jsou dále zpracovány a pro každý vrchol je vytvořen jeho herní profil jako instance třídy `Valuation`.

Vytvořené herní profily jsou porovnávány a pro všechny vrcholy hráče 0 je vybrán nejlepší následník jako nová strategie. Metoda `GetP0Strategy` pro každý vrchol porovnává příslušné herní profily a vybírá mezi nimi ten s nejlepší hodnotou, s ohledem k relevantnímu uspořádání. Vrchol s nejlepším ohodnocením se stává novou strategií zpracovávaného vrcholu. Pokud není nová strategie totožná s původní strategií, tak se nastaví příznak změny `change`. Iterace hlavního cyklu algoritmu probíhají, dokud je příznak změny nastaven.

Nebyla-li v poslední iteraci provedena žádná změna strategie, je ze strategií a herních profilů vyčteno řešení hry. Výherní regiony jsou určeny z valuace určením parity nejrelevantnějšího vrcholu. Výherní strategie každého vrcholu jsou jednoduše vyčteny z instancí `Strategy`.

5.5 Třídy pro testování

Pro část aplikace zaměřenou na testování algoritmů byly vytvořeny třídy `Benchmark` a `Stats`. Jejich hlavní úlohou je vyřešit zadané hry vybranými algoritmy a uložit údaje o řešení. Jejich vzájemný vztah je vyobrazen na Obrázku 15.

5.5.1 Třída Benchmark

Třída `Benchmark` slouží jako prostředek pro testování a porovnávání implementovaných algoritmů. Pro svou funkci si udržuje seznam her, které se mají testovat a seznam algoritmů, kterými se mají hry testovat. Po spuštění metodou `Run` pro každou hru provede otestování všemi vybranými algoritmy a uloží získané výsledky.

Při zakládání instance `Benchmark` jsou parametry konstruktoru seznam her, seznam algoritmů a počet opakování testu. Samotné testování spouští metoda `Run`. Její náplní je spustit pro každý algoritmus ze seznamu paralelně metodu `BenchmarkTask` a předat jí název algoritmu a seznam her.

Metoda `BenchmarkTask` zajišťuje spuštění konkrétního algoritmu nad všemi instancemi paritních her. Všechny hry ze seznamu vyřeší tolikrát, kolikrát je zadáno parametrem `repeatCount`. Pro získání správné instance typu `GameSolver` slouží metoda `GetSolver`, která podle názvu vytvoří správnou instanci. Pro každou iteraci `BenchmarkTask` vrací zaznamenané výsledky jako instanci třídy `Stats`.

5.5.2 Třída Stats

Instance třídy `Stats` slouží pro uchovávání výsledků generovaných při testování třídou `Benchmark`. Jejími atributy jsou velikost hry tj. počet vrcholů hry, název algoritmu a údaje o řešení. Zaznamenávané údaje o řešení jsou čas řešení, počet zpracovaných vrcholů a počet elementárních operací. Jako elementární operaci algoritmu máme například u rekurzivního algoritmu počet rekurzivních volání nebo u Small Progress Measure počet iterací hlavního cyklu. Sledovaný údaj o počtu zpracovaných vrcholů představuje celkový počet přístupů k instancím typu `Node` v průběhu řešení, jako je například přístup k atributům, následníkům atd..

Všechny údaje, kromě času, jsou instanci `Stats` předáni po vyřešení hry. Pro měření časového údaje slouží metody `startStopwatch` a `stopStopwatch`, které spouštějí a ukončují měření času výpočtu.

5.6 Implementace GUI

V této části si krátce představíme některé prvky uživatelského rozhraní. Uživatelské rozhraní zastává významnou roli ve vytvářené aplikaci. Jedním z cílů je vytvoření snadno ovladatelné a přehledné aplikace.

Část aplikace `Solver` je věnována řešení her a poskytování výsledků řešení uživateli. Kromě této funkce musí uživatelské rozhraní poskytovat možnosti jak hry načítat, generovat a ukládat. K tomuto účelu jsou využity standardní systémové dialogy. Pro snadné získání výsledku řešení, obsahuje aplikace prostor pro textový výpis řešení. Velmi dobrou pomůckou při zkoumání řešení malých her je vizualizace jejich grafu. Vizualizace automaticky zobrazí graf hry do velikosti 20 vrcholů. Pokud je k dispozici také řešení hry, příslušně vyznačí výherní regiony a strategie.

V části aplikace nazvané `Benchmark`, která je věnována testování algoritmů také nalezneme ovládací prvky pro snadné zadávání her. Společně s těmito ovládacími prvky obsahuje formulář prostor pro výběr testovacích algoritmů a ovládání zobrazených výsledků. Největší prostor zaujímá tabulka prezentující výsledky a odpovídající sloupcový graf.

5.6.1 Načítání a ukládání her a dat

K co nejjednoduššímu načítání a ukládání dat pro uživatele jsou použity komponenty `OpenFileDialog` a `SaveFileDialog`. Jejich použití vyvolá standardní rozhraní systému pro zadání cesty a názvu souboru, ze kterého chceme číst nebo do něj uložit. Pro jednoduší orientaci mezi soubory jsou uvedeným komponentám nastaveny filtry podle podporovaných přípon souborů.

Komponenty zajistí předání cesty a názvu souboru pro zpracování. Tyto údaje jsou předány instanci `ParityGame`, která zajistí další zpracování. V případě načítání hry ze souboru, načte definici hry a vytvoří její grafovou strukturu. Část aplikace `Benchmark` je doplněna o možnost nahrát více souborů her najednou. V případě ukládání her nebo výsledků, paritní hra vygeneruje soubor, jehož obsahem je buď textová definice hry, nebo její řešení. O úspěchu nebo neúspěchu požadované akce je uživatel informován.

Hru pro řešení lze do aplikace zadat také ručně. Ovládací tlačítko vyvolá zadávací formulář, kde do jednotlivých řádků může uživatel zadat definici vrcholů hry. Zadané informace o vrcholech jsou zpracovány a je vytvořena hra, kterou je potřeba předat do hlavního formuláře. Pro tyto účely jsou v zadávacích formulářích obsaženy property, které jsou naplněny daty a lze je přečíst z hlavního formuláře.

Pro možnost vygenerování náhodné hry je třeba ve formuláři zadat potřebné údaje, které jsou zpracovány instancí paritní hry. Instance `ParityGame` ze zadaných údajů vygeneruje nový graf. Část `Benchmark` obsahuje navíc možnost nechat si vygenerovat sérii náhodných her.

5.6.2 Vizualizace grafu hry a výpis řešení

Velmi užitečnou součástí aplikace je možnost nechat si zobrazit graf řešené hry. Pokud jsou aplikací řešeny malé hry, je vizualizace společně s textovým výpisem řešení, ideálním nástrojem ke sledování průběhu algoritmu. Pro textový výstup je připraveno textové pole, do nějž se vypisují všechny potřebné informace. Vizualizaci grafu hry zajišťuje knihovna Microsoft Automatic Graph Layout.[13] Jedná se o open source knihovnu poskytující rozhraní pro vytváření grafiky vrcholů, hran a zajišťující jejich optimální rozložení po zobrazovací ploše.

Zobrazení textové informace o načtené hře a jejím řešení zajišťuje komponenta `RichTextBox`. Komponenta zobrazuje všechny jí předané textové informace. Po načtení hry je to její definice. Po vyřešení hry se doplní informace o výherních regionech a strategiích. Pokud je v uživatelském rozhraní zatržena možnost výpisu postupu řešení, tak se společně s řešením vypíše i postup.

Grafické vykreslování her do formuláře provádí metoda `ShowGame`. Zobrazení grafu hry je automatické, pokud načtení hra má méně než 20 vrcholů. Při více vrcholech je nutno vyvolat vykreslení grafu ručně. Je dobré mít na paměti, že optimální vykreslení velkého množství hran u velkých her může být časově náročné.

Pro možnost vykreslení je ve formuláři komponenta `Panel`, která umožní vykreslení grafu. Do panelu se načte nová instance grafu z knihovny Microsoft Automatic Graph Layout. Vrcholy se do grafu přidávají metodou `AddEdge`, která vytváří hranu mezi vrcholy. Graf je tvořen přidáváním hran, kdy příslušné vrcholy se vytváří automaticky. Pro snadnější vyhledávání vrcholů a hran jsou vrcholy i hrany pojmenovány. Vrcholy jsou pojmenovány svým ID, které mají v grafu paritní hry. Hrany jsou pojmenovány spojením ID vrcholu, ze kterého vystupují a ID vrcholu do kterého vstupují. Vytvořeným vrcholům na obou stranách hrany přiřadíme odpovídající tvar, podle toho, který hráč jej vlastní.

Každý vytvořený vrchol grafu obsahuje informace převzaté z vrcholu paritní hry. Jedná se o prioritu, identifikátor vrcholu a textový název vrcholu, pokud existuje. V každém vrcholu vizualizovaného grafu je vepsáno číslo, případně text. Přednostně vrcholy zobrazují svou prioritu. Pro přepínání informací vepsaných ve vrcholech slouží skupina tlačítek.

Jakmile je zadaná hra vyřešena a je aktivní zobrazení grafu hry, tak informace o výherních regionech a strategiích je přenesena do vizualizace. Všem vrcholům se nastaví barva podle příslušnosti do jednoho z výherních regionů. Obarveny jsou také hrany, které jsou výherní strategií hráčů.

5.6.3 Zobrazení výsledků benchmarků

Formulář části aplikace `Benchmark`, kromě ovládacích prvků pro nastavení a spuštění, obsahuje tabulku a sloupcový graf pro zobrazení výsledků. Jsou-li výsledky testování k dispozici, jsou zobrazeny pomocí komponent `DataGridView` a `Chart`. Získané výsledky testování obsahují tři sledované parametry řešení, mezi kterými lze v zobrazení přepínat.

Data pro zobrazení v tabulce zpracovává metoda `ResultsToGridView`. Tato metoda pro každou testovanou hru zpracuje příslušné instance `Stats` a tabulku naplní získanými údaji. Pro vypsání do tabulky se tvoří instance `DataTable`, reprezentující tabulku, do které se nastaví sloupce, které má zobrazovat. Zpracováváním instancí `Stats` se vytvářejí řádky tabulky jako `DataRow`, které se přiřadí do `DataTable`. Jaké hodnoty se do tabulky načítají je řízeno volbou v uživatelském rozhraní. Při změně uživatelské volby je obsah tabulky příslušně změněn.

Zobrazení sloupcového grafu provádí metoda `ResultsToGraph`. Nejprve inicializuje oblast grafu, nastaví správné popisky a rozsahy os. Následně pro každý algoritmus a hru zprůměruje dosažené výsledky a vynese je do grafu. Každému algoritmu přiřadí jednu barvu v grafu a vytvoří odpovídající legendu. Informace zobrazovaná v grafu je řízena uživatelskou volbou společně s informacemi zobrazenými v tabulce.

6 Popis a ovládání GUI

V této kapitole bude krátce popsáno základní ovládání a vzhled vytvořené aplikace. Aplikace je vytvořena pro systém Windows a ke svému fungování potřebuje operační systém Windows 7 a novější. Pro správné spuštění je nutné mít v systému nainstalován Microsoft .NET Framework alespoň ve verzi 4.5.2. Aplikace se spouští dvojklikem na spouštěcí soubor `ParityGameSolver.exe`. Po spuštění se zobrazí uvítací okno aplikace (na Obrázku 16), které slouží jako rozcestník mezi částmi aplikace `Benchmark` a `Solver`[4].

6.1 Solver část

Část aplikace nazvaná `Solver` slouží pro vyřešení zadané paritní hry a vypsání jejího řešení. Tato část se spouští volbou `Experimenty` na uvítacím okně aplikace a její podoba je vyobrazena na Obrázku 17. Okno této části aplikace se skládá ze tří sekcí a toolbaru pro přepnutí do druhé části aplikace.

První sekce, označena na obrázku číslem 1, obsahuje ovládací tlačítka. Tlačítka `Načtení ze souboru`, `Zadání hry ručně` a `Náhodná hra` slouží pro načtení paritní hry do aplikace. Tlačítko `Načtení ze souboru` vyvolá standardní okno systému Windows pro výběr souboru s definicí paritní hry. Po úspěšném nahrání souboru dojde ke zpracování obsahu a vytvoření hry. Toto okno filtruje soubory v adresářích pro snadnější hledání souborů s hrami. Filtrované přípony souborů jsou ".pg"nebo všechny přípony.

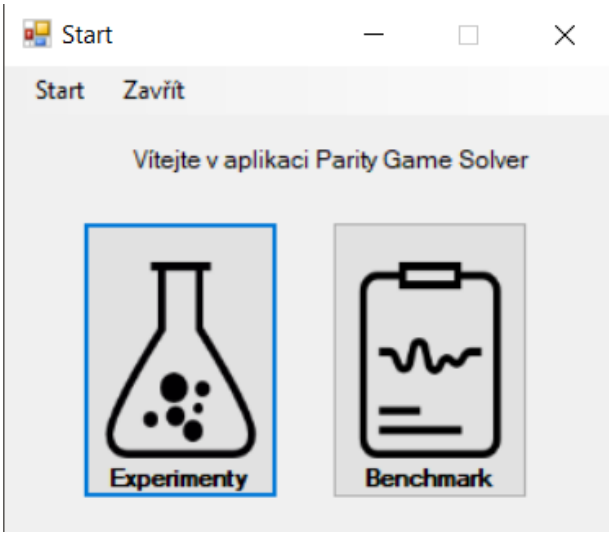
Tlačítko `Zadání hry ručně` zobrazí nové okno s tabulkou pro ruční zadání hry, které je vyobrazeno na Obrázku 18. Do řádků tabulky se zadávají informace o vrcholech hry. Po vyplnění a potvrzení tlačítkem `Hotovo` se aplikace pokusí vytvořit zadanou hru. Vyskytne-li se při vytváření hry chyba, je uživatel vyzván k opravě chybně zadaného řádku. Pokud byla v aplikaci aktuálně načtena nějaké hra, tak je tabulka již předvyplněna. Předvyplněnou tabulku lze upravit nebo zcela vymazat.

Pro vytvoření náhodné paritní hry slouží tlačítko `Náhodná hra`. Stisknutím tlačítka je vyvoláno okno na Obrázku 19. Jsou-li parametry zadané v odpovídajících mezích, vygeneruje se náhodná paritní hra. V opačném případě je uživatel vyzván k opravě chybně zadaného parametru.

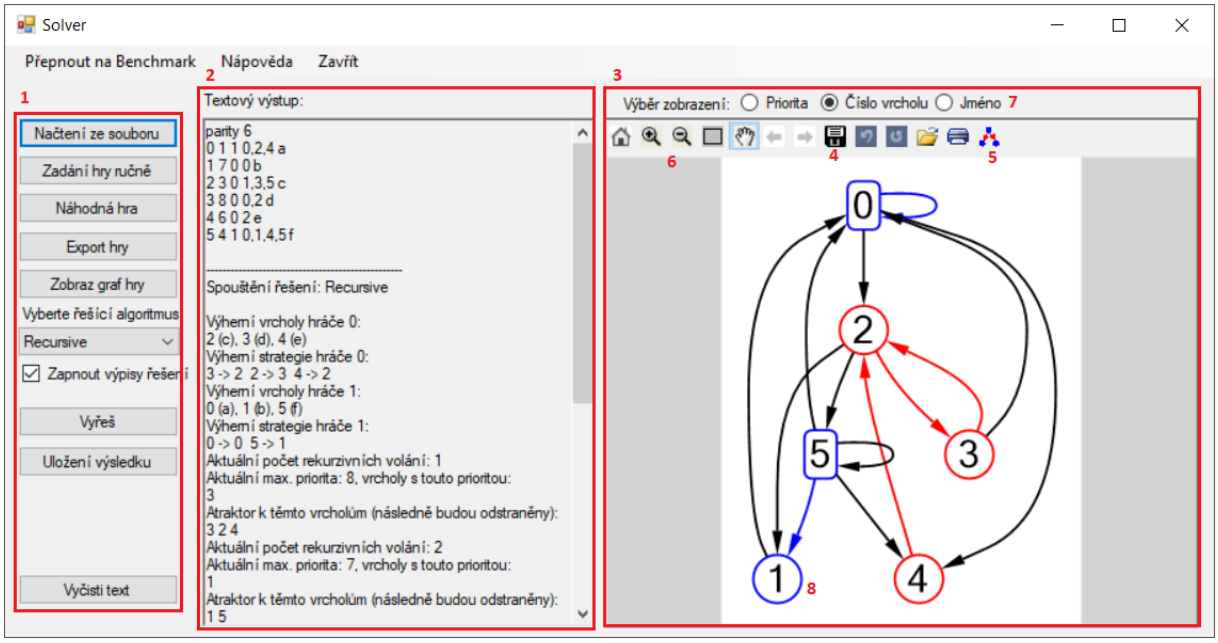
Pro export hry načtené v aplikaci je připraveno tlačítko `Export hry` a pro export řešení hry tlačítko `Uložení výsledku`. Obě tlačítka vyvolají standardní okno systému Windows pro uložení souboru. Export probíhá ve stejném formátu, v jakém hry načítáme.

Výběrové menu se seznamem algoritmů určuje, jakým algoritmem se načtená paritní hra bude řešit. Samotné řešení se spouští tlačítkem `Vyřeš`. Před spuštěním řešení lze zatrhnutím volby `Zapnout výpisy řešení` ovlivnit, zda se v textovém výpisu budou zobrazovat informace o průběhu řešení nebo pouze samotné řešení.

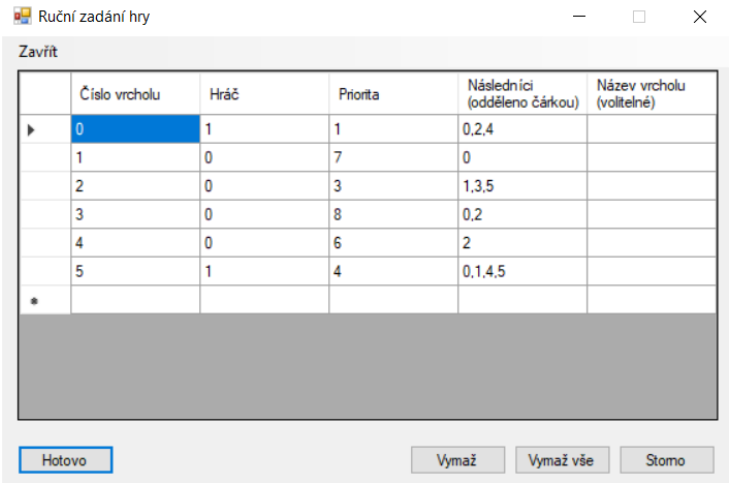
[4] Volba Experimenty.



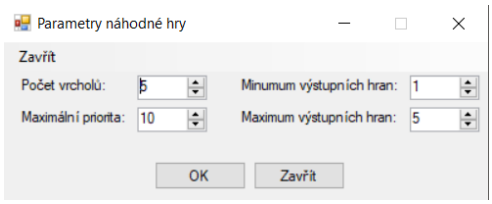
Obrázek 16: Uvítací okno aplikace



Obrázek 17: Hlavní okno části Solver



Obrázek 18: Formulář pro ruční zadání hry

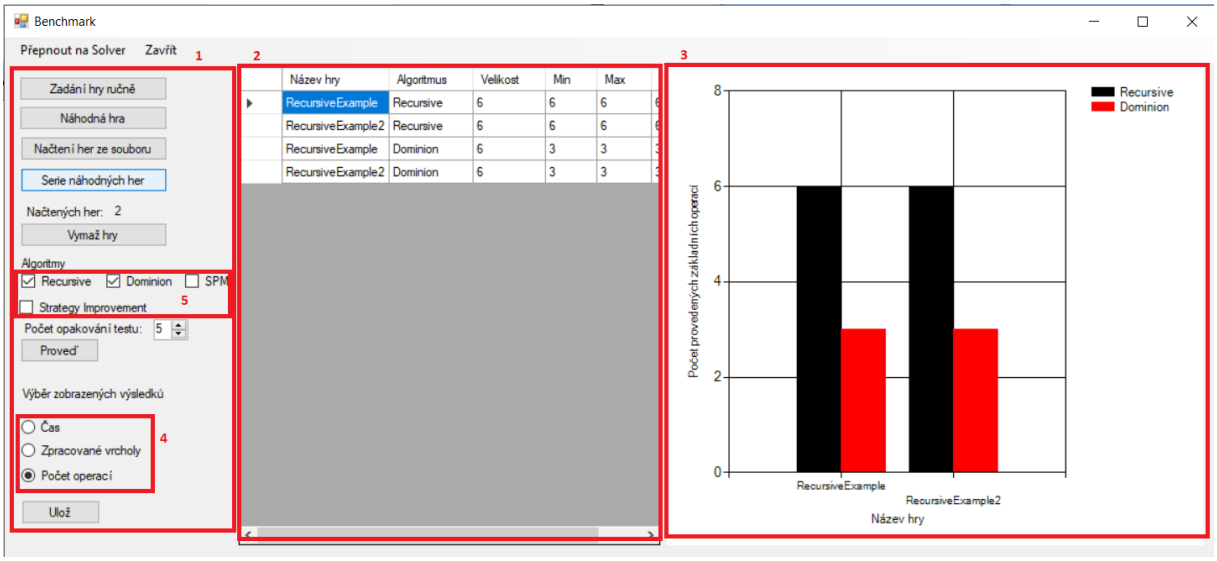


Obrázek 19: Okno pro nastavení parametrů náhodné hry

Druhá sekce, označena na obrázku číslem 2, je textové pole. Toto pole slouží pro výpis definice načtené hry, zobrazení výsledného řešení hry a případně průběhu řešení. Textové pole není editovatelné a lze v případě potřeby vymazat tlačítkem `Vyčisti text`.

Třetí sekce, označena na obrázku číslem 3, je prostor pro zobrazení grafu načtené hry. Grafické zobrazení probíhá automaticky do velikosti hry 20 vrcholů. Automatické vykreslování grafu je omezeno jak z důvodu časové náročnosti vykreslení, tak z důvodu nepřehlednosti grafu s vysokým počtem vrcholů a hran. Pokud si i přesto uživatel přeje zobrazit graf, může zobrazení vyvolat tlačítkem `Zobraz graf hry`. Je-li zobrazení aktivní a je-li k dispozici řešení zobrazené hry, tak jsou v grafu barevně vyznačeny výherní regiony a strategie. V každém vyobrazeném vrcholu je vepsána informace o vrcholu. Přepínání informací, které jsou ve vrcholu vepsány provádí skupina radio buttonů označených číslem 7. Vrchol zobrazující svůj identifikátor je na obrázku označen číslem 8.

Zobrazovací plocha obsahuje panel nástrojů nabízející funkce vztahující se k vizualizaci grafu. Panel nabízí možnost uložení zobrazeného grafu bitmapově i vektorově. Tato možnost je označena v obrázku číslem 4. Volba označena v obrázku číslem 5 umožňuje volit mezi několika způsoby generování vizualizace grafu. Tlačítka označena číslem 6 provádí přibližování a oddalování grafu. Graf lze také přibližovat a oddalovat kolečkem myši. Zobrazovací plocha grafu neobsahuje posuvníky a posuny zobrazení jsou řešeny nástrojem ruka. Kliknutím a podržením levého tlačítka myši na zobrazované ploše můžeme plochou pohybovat. Pokud je zobrazovací plocha malá,



Obrázek 20: Okno benchmark pro testování

dvojklikem do zobrazovací plochy se okno maximalizuje.

6.2 Benchmark část

Podoba části aplikace nazvané `Benchmark` je na Obrázku 20. Tato část slouží pro porovnávání algoritmů a skládá se ze tří částí.

První část, na obrázku označena číslem 1, je složena z ovládacích prvků. Tlačítka `Zadání hry ručně`, `Náhodná Hra` a `Načtení her ze souboru` fungují obdobně jako v případě části `Solver`. Rozdílem je, že načtené nebo vygenerované hry se přidávají do seznamu her k řešení. U načítání her ze souboru lze v této části zároveň načíst několik definic her najednou. Oproti části `Solver` přibylo tlačítko `Série náhodných her`. Toto tlačítko vyvolá formulář, který po zadání parametrů vygeneruje několik náhodných her o zvětšujícím se počtu vrcholů. Všechny hry načtené v aplikaci lze tlačítkem `Vymaž hry` smazat.

Skupina políček, označených číslem 5, vybírá kterými algoritmy budou hry vyřešeny. Pod touto skupinou lze navolit počet opakování každého výpočtu. Stisknutím tlačítka `Proveď` se začne testování.

Druhá část, na obrázku označena číslem 2, obsahuje tabulku zobrazující výsledek testování. Tabulka má sloupce zobrazující název řešené hry, algoritmus kterým byla hra řešena, velikost hry a minimální, maximální a průměrnou naměřenou hodnotu. Každý řádek tabulky obsahuje zpracované výsledky všech opakování řešení jedné hry, jedním algoritmem. Například první řádek obsahuje výsledky pěti iterací rekurzivního algoritmu na hře pojmenované RecursiveExample. Sloupce Min, Max a Avg obsahují minimální, maximální a průměrnou hodnotu výsledku z těchto pěti iterací. Informace, které lze zobrazit jsou doba zpracování, počet zpracovaných vrcholů a

počet provedených základních operací. Mezi těmito informacemi lze přepínat. K přepínání slouží přepínače označené na obrázku číslem 4.

Třetí část, na obrázku označena číslem 3, zobrazuje formou sloupcového grafu výsledky z tabulky. Každému algoritmu je přiřazena barva, kterou je v grafu vyobrazen. Jednotlivé sloupce grafu odpovídají řádkům v tabulce a zobrazují hodnotu sloupce Avg. V grafu jsou zobrazovány údaje podle přepínače, stejně jako v případě tabulky. Změnou přepínače se změní údaje jak v tabulce, tak v grafu. Zobrazovací plocha grafu se dá zvětšit dvojklikem do plochy grafu.

Získané výsledky testování lze exportovat ve formátu csv do souboru pomocí tlačítka `Ulož`.

7 Testování a srovnání výkonnosti algoritmů

Pro potřeby testování bylo vytvořeno několik sad paritních her sledujících chování algoritmů za určitých podmínek. Při testech sledujeme chování algoritmů na hrách, u kterých zvyšujeme počet vrcholů, maximální možnou prioritu anebo maximum odchozích hran. Údajem sledovaným při testování je primárně doba trvání výpočtu. Dalšími sledovanými hodnotami jsou počet navštívených vrcholů a počet provedení hlavní operace algoritmu. Počet navštívených vrcholů znamená počet přístupů k informacím o vrcholech v průběhu výpočtu. Jedná se například o zjišťování následníků vrcholu nebo jeho atributů. Hlavní operací u rekurzivního algoritmu a hledání malých dominií je rekurzivní volání nebo iterace hlavního cyklu u small progress measures a strategy improvement.

První testovaná sada obsahuje dvanáct náhodných her se zvyšujícím se počtem vrcholů. Tyto hry reprezentují nejčastěji používaný typ her při testování. Na druhé sadě her sledujeme chování algoritmů za situace, kdy je zafixován počet vrcholů hry a mění se maximální priorita. Tato sada obsahuje deset her. Poslední sada obsahuje také deset her, u kterých se zvyšuje maximální počet odchozích hran při fixním počtu vrcholů. Mimo sady náhodných her jsou do testování zahrnuty speciální typy her, které jsou navrženy tak, aby byly obtížné pro určitý algoritmus.

Každý algoritmus byl na každé hře testován sedmkrát, aby se minimalizovala nepřesnost. Z každého testování byly odstraněny výsledky prvního testu. Ze zbylých výsledků je brána průměrná hodnota. Výsledky prvního testování jsou odstraňovány z důvodu odstranění vlivu just in time kompilace v průběhu prvního výpočtu.

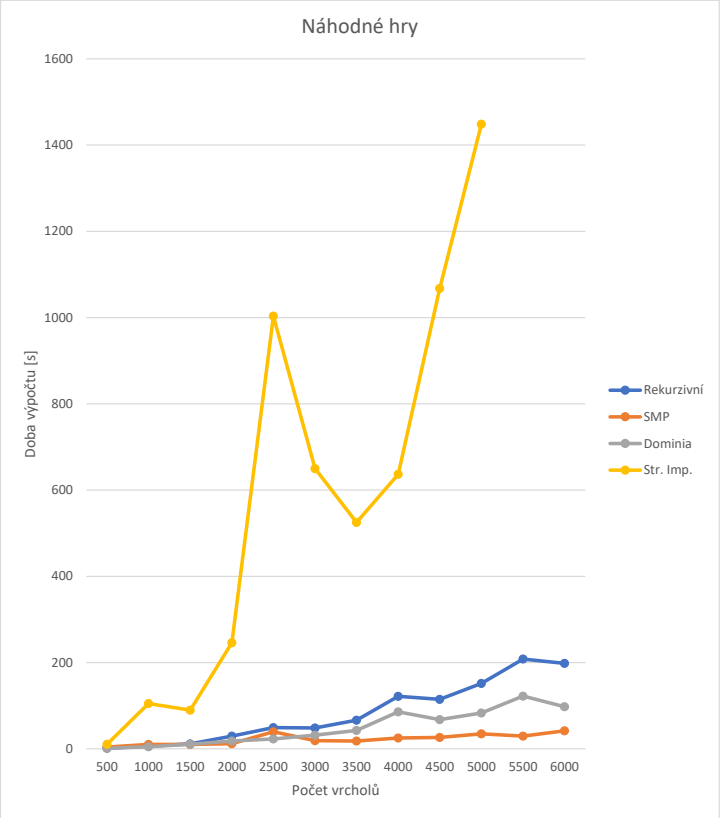
Zpracované výsledky měření jsou vyneseny do grafů pro snadné porovnání. V grafech nejsou u jednotlivých bodů vyznačeny přesné hodnoty získané testováním, protože nejsou primárně důležité. Důležité z pohledu testování je sledování růstu hodnot mezi jednotlivými sadami her u jednotlivých algoritmů a porovnání průběhů grafů algoritmů mezi sebou.

7.1 Náhodné hry

7.1.1 Zvyšující se počet vrcholů

Pro tento typ testu bylo vygenerováno dvanáct paritních her, které se liší svým počtem vrcholů. Nejmenší hra obsahuje 500 vrcholů a největší obsahuje 6000 vrcholů. Každá následující hra je tedy o 500 vrcholů větší. Z výsledků testování byly vytvořeny dva grafy. První graf na Obrázku 21 obsahuje porovnání doby výpočtu jednotlivých algoritmů. Druhý graf na Obrázku 22 srovnává počty zpracovaných vrcholů. Výsledky uvádějící počet provedených základních operací nejsou uvedeny z důvodu, že pro všechny algoritmy byly hodnoty téměř totožné.

U rekurzivního algoritmu, hledání malých dominií a small progress measuress jsou výsledky měření předvídatelné. Jako základ srovnání poslouží rekurzivní algoritmus. Jak v časovém měření, tak v počtu navštívených vrcholů jsou si průběhy výsledků velmi podobné. Hledání malých dominií dosahuje přibližně dvakrát kratší doby zpracování než rekurzivní algoritmus. To



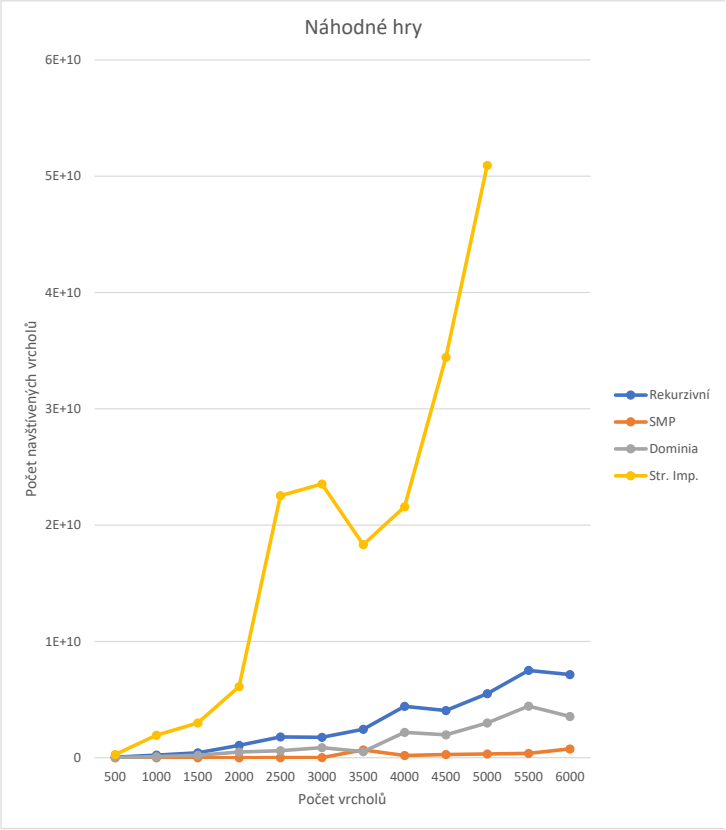
Obrázek 21: Porovnání času výpočtu na náhodných hrách

dokazuje, že na náhodných hrách přináší zlepšení výpočetního času, jak bylo předpokládáno. Nejrychlejším algoritmem byl small progress measures, který předčil rekurzivní zhruba třikrát. Tyto výsledky odpovídají předpokládanému chování algoritmů vůči jejich popsané složitosti.

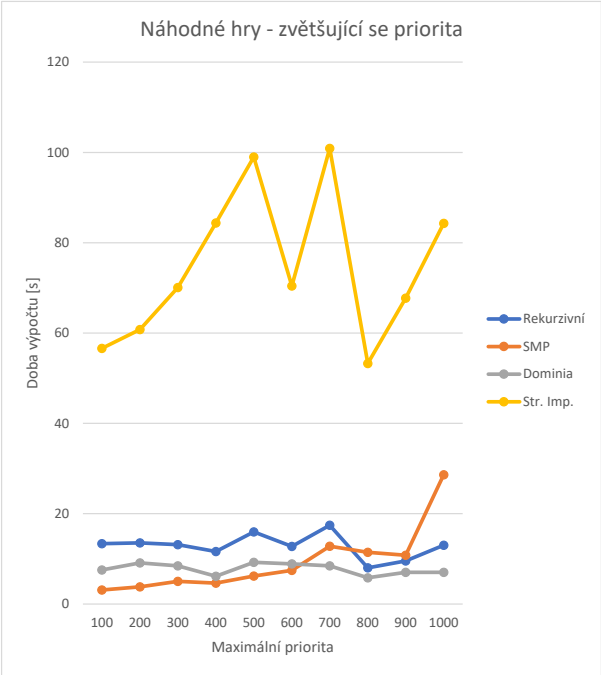
Překvapivé výsledky přinesl algoritmus strategy improvement. Už na malých hrách je k vyřešení hry potřeba mnohonásobně více času a navštívených vrcholů než u ostatních algoritmů. U dvou posledních testovacích her o velikosti 5500 a 6000 vrcholů se nepodařilo hry vyřešit v časovém limitu 30 minut a měřené hodnoty nebyly zaznamenány. Lze předpokládat, že za neuspokojivými výsledky stojí nutnost generování podher v každé iteraci.

7.1.2 Zvyšující se maximální priorita vrcholů

Tento test se zabývá vlivem změny maximální přípustné priority vrcholů na složitost výpočtu. Pro tyto účely bylo vygenerováno 10 her, kde je počet vrcholů nastaven na 1500 a počet odchozích hran mezi 1-200. Měnícím se prvkem je maximální možná priorita vrcholů. Ta je nastavována v rozmezí sto až tisíc. Přitom minimální priorita zůstává u všech her nastavena na 0. Výsledky testů jsou zobrazeny na Obrázku 23 a Obrázku 24. První z grafů ukazuje změny času potřebného k vyřešení her se zvyšující se prioritou a druhý počet navštívených vrcholů.



Obrázek 22: Porovnání počtu zpracovaných vrcholů při výpočtu na náhodných hrách



Obrázek 23: Porovnání času výpočtu na náhodných hrách s rostoucí prioritou

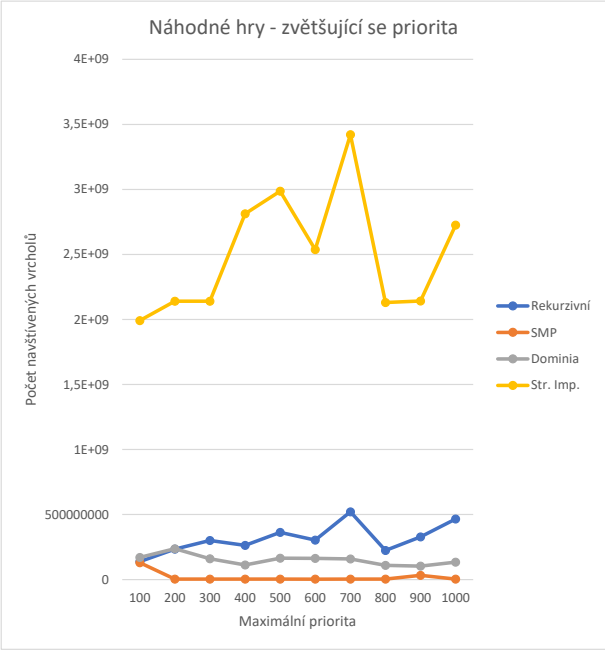
Rekurzivní algoritmus a hledání malých dominií vykazují pouze malé změny času výpočtu se změnou priorit. A obdobné zjištění lze sledovat u počtu zpracovaných vrcholů. U small progress measures můžeme pozorovat postupně se zvyšující dobu trvání. Pravděpodobná příčina je ve velikosti porovnávaných vektorů jednotlivých vrcholů. Se zvyšujícím se počtem priorit ve hře roste také velikost small progress measure vektorů, které je nutno porovnávat. K tomuto zjištění přispívá skutečnost, že u počtu navštívených vrcholů neroste průběh grafu stejně.

Nejhorší výsledky jsou zaznamenány u strategy improvementu. Jeho časy potřebné k výpočtu i navštívené vrcholy několikanásobně převyšují ostatní algoritmy. Lze se domnívat, že časy u jednotlivých her nejsou zcela ovlivněny zvyšující se prioritou, ale spíše charakterem náhodných her. Tato domněnka plyne z pozorování, že měřené hodnoty zůstávají v určitých mezích i při zvyšující se prioritě, na rozdíl od small progress measures.

7.1.3 Zvyšující se maximální počet odchozích hran z vrcholu

Pro otestování vlivu počtu odchozích hran bylo také vygenerováno deset her. Všechny hry se skládají z 1500 vrcholů a jejich maximální priorita je nastavena na 200. Měnícím se parametrem her je maximum možných odchozích hran. Počet odchozích hran se mění v rozmezí 100 až 1000 a krok zvětšování je 100 hran. Naměřené hodnoty jsou vyneseny v grafech na Obrázku 25 a Obrázku 26.

U testování závislosti složitosti výpočtu na počtu odchozích hran je na rozdíl od předchozího testu vidět zvyšující se složitost u všech algoritmů. Z výsledků plyne, že nejlepším algoritmem je i v tomto případě small progress measures, následovaný hledáním malých dominií a rekurzivním



Obrázek 24: Porovnání počtu zpracovaných vrcholů při výpočtu na náhodných hrách s rostoucí prioritou

algoritmem. Jako nejhorší se opět projevil strategy improvement. U rekurzivního algoritmu zaznamenáváme pětkrát delší čas výpočtu i navštívených vrcholů u hry s maximem 1000 odchozích hran, oproti hře se 100 odchozími hranami. Trojnásobný nárůst je zaznamenán také u hledání malých dominií. U small progress measures bylo zhoršení času výpočtu asi desetinásobné. Tyto údaje jsou zobrazeny v detailu na Obrázku 27. Výsledky u rekurzivního algoritmu a hledání malých dominií naznačují zvýšenou obtížnost hledání atraktorů při řešení. U small progress measures je pak třeba porovnávat větší množství vektorů.

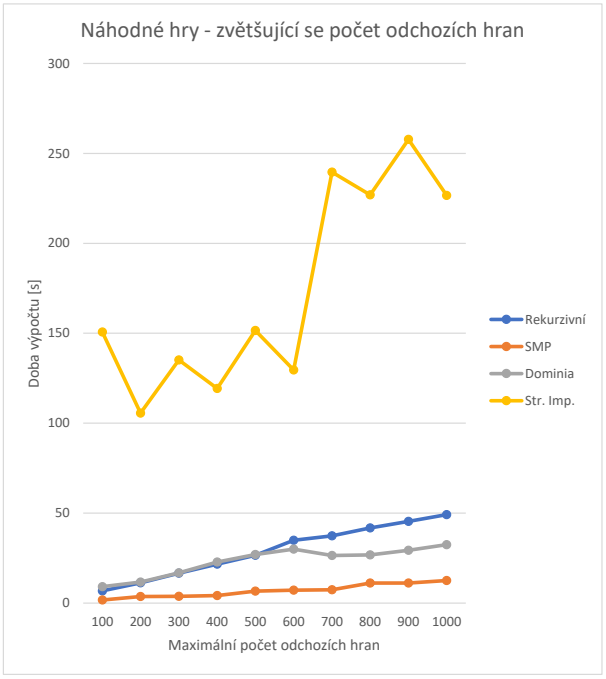
Strategy improvement, který z testu vyšel nejhůře vykazuje podobně strmé zvýšení sledovaných hodnot při porovnání s detailem ostatních her. Zhoršení bylo přibližně trojnásobné, což je srovnatelné s hledáním malých dominií. Nicméně doba výpočtu značně přesahuje zbylé algoritmy.

7.2 Speciální hry

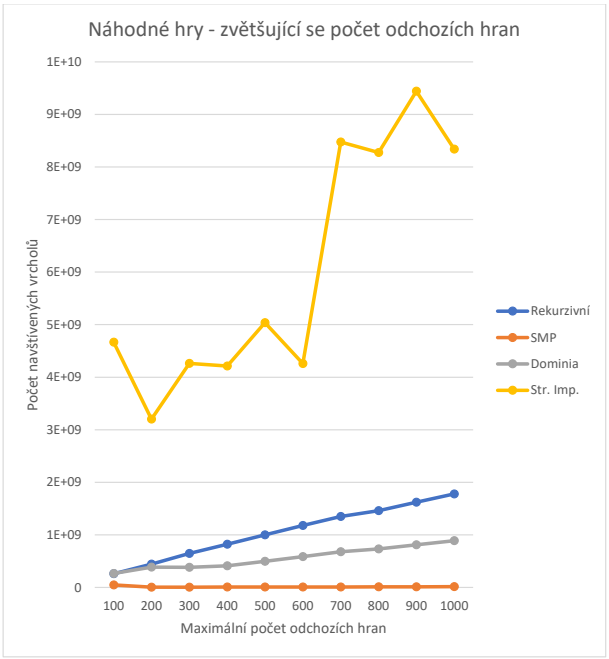
7.2.1 Recursive ladder

Recursive ladder je typ her speciálně navržených pro rekurzivní algoritmus. Testované hry byly vygenerované v programu PGSolver. Bylo vygenerováno 6 her, které byly otestovány rekurzivním algoritmem a pomocí strategy improvement. Výsledné časy testů jsou v grafu na Obrázku 28. Jednotlivé hry jsou pojmenovány svým číslem, které značí míru rekurze.

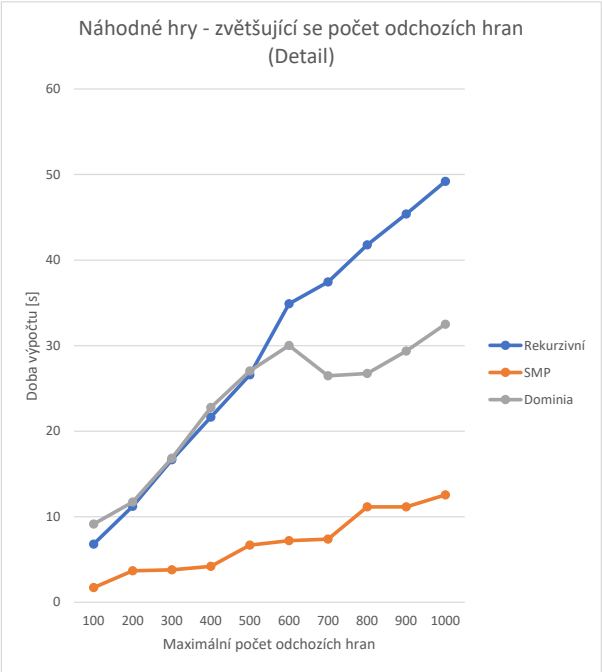
Z grafu lze vidět, že už u malých her vykazuje rekurzivní algoritmus vysoký nárůst výpočetního času. U páté testované hry již došlo k přesažení třicetiminutového limitu. Pro porovnání



Obrázek 25: Porovnání času výpočtu na náhodných hrách s rostoucím počtem odchozích hran



Obrázek 26: Porovnání počtu zpracovaných vrcholů při výpočtu na náhodných hrách s rostoucím počtem odchozích hran

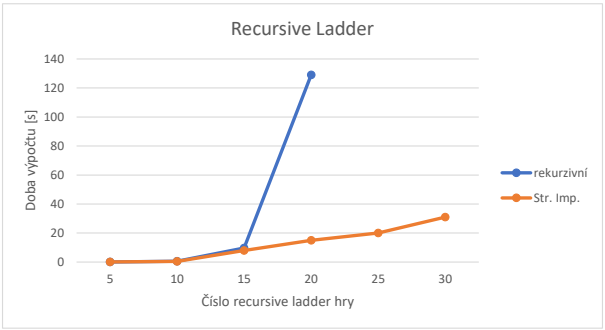


Obrázek 27: Detail porovnání času výpočtu na náhodných hrách s rostoucím počtem odchozích hran

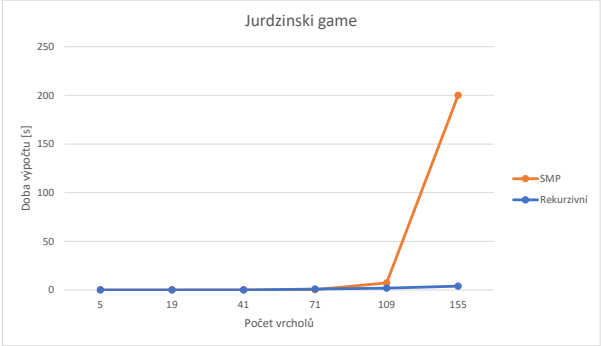
je uveden strategy improvement, který s tímto typem her nevykazoval tak značný nárůst doby výpočtu.

7.2.2 Jurdziński game

Pro otestování exponenciálního chování algoritmu small progress measures vznikl speciální typ her nazvaných jako Jurdziński games. Pomocí PGSolveru bylo vygenerováno 6 her. Jedná se o grafy velikosti $(2d + 1) \times (2w)$, kde d a w jsou zadávané parametry. Graf hry se tak rychle zvětšuje. Všechny hry byly vyřešeny rekurzivním algoritmem a algoritmem small progress measures. Naměřené hodnoty jsou na Obrázku 29. Nejmenší testovaná hra obsahuje pouze 5 vrcholů a největší 155 vrcholů.



Obrázek 28: Srovnání na hrách typu Recursive ladder



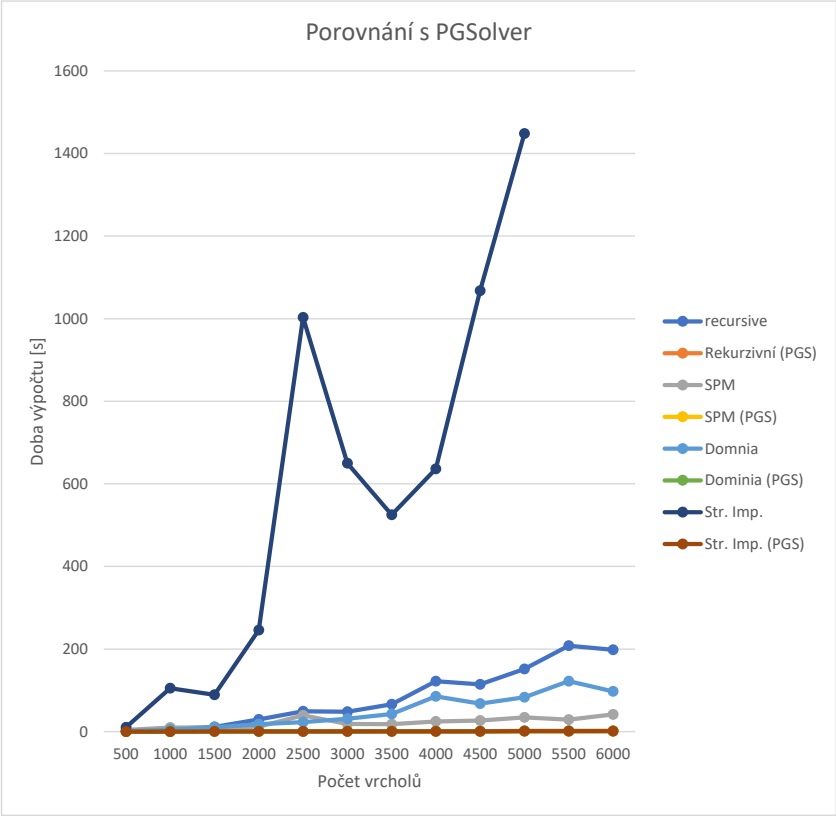
Obrázek 29: Srovnání na hrách typu Jurdzinski game

Z naměřených výsledků je možno konstatovat, že již při malém počtu vrcholů (155) dosahuje small progress measures enormní časové náročnosti. Zatímco rekurzivní algoritmus zvládl poslední dvě hry vyřešit v jednotkách sekund, tak small progress measuress mnohonásobně tyto časy překračuje. Případná další vygenerovaná hra tohoto typu obsahuje 209 vrcholů a řešení přesáhne časový limit.

7.3 Srovnání s PGSolverem

V poslední části srovnáme vzniklou implementaci s programem PGSolver. Pro srovnání budou sloužit stejné náhodné hry jako u prvního testování. Jedná se tedy o náhodné hry o velikosti 500 až 6000 vrcholů. Výsledné srovnání je zobrazeno na Obrázku 30.

Z porovnání výsledků vyplývá, že implementace algoritmů v programu PGSolver vykazuje lepší výpočetní časy. Všechny hry byly PGSolverem vyřešeny v jednotkách vteřin. Tento výsledek je nejspíše dán řadou optimalizací, které PGSolver používá. Vliv může mít i použití jiného prostředí, programovacího jazyka a datových struktur používaných při reprezentaci a práci s grafem her.



Obrázek 30: Srovnání algoritmů s programem PGSolver

8 Závěr

Diplomová práce se zabývá problematikou paritních her, které jsou i přes svou jednoduchost stále obtížně řešitelné. První část práce se zabývá uvedením čtenáře do kontextu paritních her. Jsou představeny základní vlastnosti paritních her a jejich struktura. Popis her, jejich struktury a navazujících pojmů se neomezuje pouze na strohý výčet definic, ale snaží se problematiku čtenáři co možná nejvíce přiblížit a vysvětlit za současného zachování dostatečné míry detailnosti a formálnosti.

V podobném konceptu se nesou definice a popis algoritmů, jež byly v této práci zkoumány. Celkem byly zpracovány čtyři algoritmy zastupující všechny hlavní přístupy k řešení paritních her. Základní a nejstarší způsob řešení paritních her je rekurze. Do této kategorie spadají dva algoritmy. Jedná se o rekurzivní algoritmus a hledání malých dominií. Rekurzivní algoritmus reprezentuje přímočarý způsob řešení, zatímco hledání malých dominií ukazuje možnosti, jak je možné vylepšit princip rekurzivního algoritmu vhodným předzpracováním grafu hry. Zbylé přístupy nejsou založeny na rekurzivním zpracování podher, nýbrž spoléhají na iterativní hledání řešení. Small progress measures zastupuje algoritmy, které vrcholům přiřadí hodnotící vektor, jehož hodnota je upravována vůči vektorům sousedních vrcholů tak dlouho, dokud není dosaženo rovnovážného stavu. Poslední reprezentant zastupuje algoritmy zlepšující strategie. Algoritmus discreate strategy improvement iterativně mění strategie vrcholů jednoho hráče až do okamžiku, kdy žádná jiná strategie neposkytuje zlepšení.

Tyto algoritmy byly vybrány, protože se jevily jako ideální zástupci všech čtyři přístupů. Při popisu jednotlivých algoritmů byl kladen důraz jak na přesnost definic, tak na co největší srozumitelnost. Pro snadné pochopení všech popsaných principů je u každého algoritmu uveden detailní rozbor řešení jedné malé hry.

Druhá část diplomové práce se zabývá návrhem a implementací aplikace, která umožňuje řešení a testování paritních her. Podobný softwarový nástroj již existuje a nazývá se PGSolver. Návrhu vlastního softwarového řešení tak předchází krátké shrnutí PGSolveru a uvedení textového formátu, který používá k ukládání her.

Stěžejním přínosem této diplomové práce je vytvořená aplikace. Při návrhu bylo nutné se vypořádat s výběrem programovacího jazyka, prostředí a nalézt vhodné datové struktury. Mezi základní požadavky na vyvíjenou aplikaci patřila jednoduchost ovládání pro uživatele a snadná rozšiřitelnost o případné nové algoritmy nebo úpravy stávajících. Jedním z podstatných bodů při návrhu byl také formát dat pro import a export. Výsledná aplikace splňuje všechny kladené cíle. Zvolením klasické oknové aplikace bylo dosaženo jednoduchosti ovládání a zvolenou strukturou dat byla zajištěna snadná rozšiřitelnost. Výsledný kód v jazyce C# je přehledný a snadný na pochopení, což usnadňuje případné modifikace.

Všechny algoritmy obsažené v aplikaci používají pro svou funkci jednotné rozhraní, které zaručuje jejich shodné použití v rámci aplikace. U rekurzivního algoritmu a hledání malých dominií je co nejvíce využíváno pomocných metod objektu paritní hry. Bohužel tento přístup

nebylo možno využít u zbývajících algoritmů, které jsou svým principem fungování zcela odlišné. Dalším přínosem aplikace oproti jiným nástrojům je možnost vizualizace. Vizualizace grafu při řešení malých her je skvělou pomůckou pro uživatele, který se seznamuje s problematikou paritních her a fungováním jednotlivých algoritmů. Obdobně užitečné je i jednoduché zanesení výsledků testování algoritmů do grafu pro jejich rychlé porovnání.

Závěrečná část práce je zaměřena na otestování algoritmů implementovaných ve vyvinuté aplikaci. Testování proběhlo na několika sadách náhodných her, které testovaly vliv parametrů hry na složitost výpočtu. Pro otestování exponenciální složitosti algoritmů byly představeny dva druhy speciálních her. Tyto hry nejsou náhodné, ale jsou generovány pro tyto účely. Posledním bodem testování bylo porovnání s programem PGSolver.

Při testování na náhodných hrách se ukázalo, že nejlepší algoritmus je small progress measures, který předčil ostatní algoritmy ve všech testech. Jako nejhorší se ve všech případech ukázal strategy improvement, který na rozdíl od zbylých algoritmů vytváří velké množství podher zpomalujících řešení. Při porovnání rekurzivního algoritmu a hledání malých dominií bylo zjištěno, že vykazují předpokládané chování. Předpokladem bylo, že hledání malých dominií bude vždy lepší než rekurzivní algoritmus z důvodu, že je v principu jeho vylepšenou verzí.

Na dvou speciálních typech her bylo předvedeno exponenciální chování rekurzivního algoritmu a algoritmu small progress measures. Oba dva algoritmy již při hrách o malém počtu vrcholů překročili stanovený časový limit výpočtu. Bylo tím ukázáno, že výkon algoritmu je silně závislý na grafu řešené hry.

Poslední částí testování bylo porovnání vzniklé implementace s programem PGSolver. Z tohoto porovnání vyšel lépe konkurenční program. PGSolver byl lepší ve všech testovaných částech. Po porovnání výsledků se jeví jako dobrý námět na pokračování práce začlenění různých typů předzpracování grafu a optimalizací zlepšující časy řešení. PGSolver množství různých optimalizací nabízí.

Diplomová práce jako celek nabízí pohled do problematiky paritních her. Práce může sloužit jako materiál pro další zkoumání paritních her a přináší nástroj, který slouží jak pro testování a řešení her, ale také jako pomůcka při snaze pochopit implementované algoritmy.

Literatura

[1] Erich GRÄDEL, Wolfgang THOMAS a Thomas WILKE (eds.). Automata logics, and infinite games. New York: Springer, 2002. ISBN 3-540-00388-6.

[2] Adam ANTONIK, Nathaniel CHARLTON, Michael HUTH. Polynomial-time underapproximation of winning regions in parity games. In *Electronic Notes in Theoretical Computer Science*, 2009, vol. 225, p. 115-139. ISSN: 1571-0661.

[3] Oliver FRIEDMANN. Recursive algorithm for parity games requires exponential time. In *RAIRO - Theoretical Informatics and Applications*, 2011, vol. 45, no. 4, p. 449-457. DOI:10.1051/ita/2011124

[4] Marcin JURDZIŃSKI. Deciding the winner in parity games is in UP ∩ `co-UP`. In *Information Processing Letters*. 1998, vol. 68, no. 3, p. 119-124. DOI: 10.1016/S0020-0190(98)00150-1.

[5] Wieslaw ZIELONKA. Infinite games on finitely coloured graphs with applications to automata on infinite trees. In *Theoretical Computer Science*. 1998, vol. 200 no.1-2, p. 135-183. DOI: 10.1016/S0304-3975(98)00009-7.

[6] Marcin JURDZIŃSKI, Mike PATERSON, Uri ZWICK. A Deterministic Subexponential Algorithm for Solving Parity Games. In *SIAM Journal on Computing*. 2008, vol. 38, p. 1519-1532. DOI: 10.1137/070686652

[7] Krzysztof R. APT, Erich GRÄDEL (ed.). *Lectures in game theory for computer scientists.* Cambridge: Cambridge University Press, 2011. ISBN: 978-0-521-19866-0.

[8] Marcin JURDZIŃSKI. Small Progress Measures for Solving Parity Games. In *STACS 2000: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, 2000, p. 290–301. DOI: 10.1007/3-540-46541-3_24.

[9] Jens VÖGE, Marcin JURDZIŃSKI. A discrete strategy improvement algorithm for solving parity games. In *Computer Aided Verifcation*. 2000, p. 202-215. DOI: 10.1007/10722167_18.

[10] Jan OBDRŽÁLEK. *Algorithmic Analysis of Parity Games: dizertační práce*. Edinburgh: University of Edinburgh, 2006.

[11] Oliver FRIEDMANN, Martin LANGE. The PGSolver Collection of Parity Game Solvers, version 3: report. Mnichov: Institut für Informatik, Ludwig-Maximilians-Universität München, 2014.

[12] Introduction to the C# Language and the .NET Framework. [online]. [cit. 2016-03-07]. <https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx>

[13] Microsoft Automatic Graph Layout. [online]. [cit. 2016-03-07]. <http://research.microsoft.com/en-us/projects/msagl/>

[14] Jeroen KAIREN. Benchmarks for Parity Games. In *Fundamentals of Software Engineering*. 2015, p. 127-142. DOI: 10.1007/978-3-319-24644-4_9.